Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

HERRAMIENTAS PYTHON PARA LA PREDICCIÓN DE ENERGÍAS RENOVABLES

Autor: Juan Bella Santos
Tutor: José Ramón Dorronsoro Ibero

ENERO 2018

HERRAMIENTAS PYTHON PARA LA PREDICCIÓN DE ENERGÍAS RENOVABLES

Autor: Juan Bella Santos
Tutor: José Ramón Dorronsoro Ibero

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
ENERO 2018

Resumen

Resumen

Las energías renovables cada vez tienen más importancia por lo que las compañías eléctricas precisan saber cuánta energía van a generar a partir de estas para que, junto a la energía obtenida de manera tradicional, poder ofrecer en el mercado la potencia total producida de forma ajustada. El problema de las energías renovables es que son muy irregulares, por lo que no hay forma segura de conocer la energía que se va a conseguir mediante ellas. Una forma de solucionar esto es mediante la aplicación de aprendizaje automático sobre los datos meteorológicos y las producciones energéticas asociadas a ellos para poder predecir la energía que se va a generar. En este trabajo se estudia cómo aplicar esta solución mediante el lenguaje de programación Python. Para ello se ha creado un sistema de recepción, tratado y procesado de datos lo más fiel a la realidad posible.

Primero se han estudiado dos tipos de ficheros altamente utilizados para el guardado de datos meteorológicos, los formatos GRIB y netCDF. De estos ficheros hemos estudiado su estructura y cómo obtener los datos contenidos en ellos mediante Python, específicamente mediante las librerías `pygrib` y `netCDF4`. Con los datos ya extraídos, se ha utilizado la librería `pandas` para organizar los datos en forma de `Dataframe`, para posteriormente poder guardarlos de forma sencilla y eficaz mediante el formato `pickle`.

Finalmente, para la realización de las predicciones energéticas, se han estudiado las bases del aprendizaje automático y la aplicación de las mismas en Python mediante la librería `sklearn`. Dentro de este tema, nos hemos centrado en los modelos lineales, como mínimos cuadrados o ridge.

Palabras Clave

Aprendizaje automático, energía eólica, meteorología, Python, Dataframe, sklearn, pygrib, netCDF4.

Abstract

Nowadays, renewable energies are gaining more importance, so electric companies need to know how much energy are going to generate from them in order to, in conjunction with the energy obtained in a more traditional way, be able to offer in the market an adjusted value of the total electric power produced. The problem with renewable energies is that they are very irregular, so there is no certain way to know how much energy it is going to be obtained from them. One solution for this problem is through the application of machine learning to the meteorological data and to the energy produced in these weather conditions to predict how much energy is going to be produced. In this assignment it is studied how to apply this solution using the Python programming language. To do this, we have created a system of reception, treatise and data processing as similar to a real system as possible.

First of all, we have studied two types of files highly used to save meteorological data, these are GRIB and netCDF. From these files, we have studied their structure and how to obtain the data contained in them using Python, specifically with the `pygrib` and `netCDF4` libraries. Once the data were extracted, we have used the `pandas` library to organize the data in `Dataframes` and save them in an easy and efficient way in `pickle` format.

Finally, to do the energy predictions, we have studied the basics of machine learning and the application of them in Python with the `sklearn` library. Inside of this topic, we have focused on linear models, like least squares and ridge regression.

Key words

Machine learning, wind power, meteorology, Python, Dataframe, sklearn, pygrib, netCDF4.

Agradecimientos

Quisiera agradecer al IIC y en particular a Julia y a José la oportunidad de realizar este trabajo y el apoyo y ayuda recibidos en todo momento.

También me gustaría agradecérselo a mis compañeros de clase, sin los cuales estos últimos años no hubiesen sido lo mismo, tanto social como académicamente.

Sin ninguna duda quiero dar las gracias a mis padres, abuelos, tíos y primos por estar siempre ahí, ayudándome, apoyándome y sobre todo soportándome en las épocas de estrés.

Finalmente, pero no por ello menos importantes, gracias a todos esos amigos que me han ayudado a desconectar de todo y centrarme en divertirme y pasar grandes ratos con ellos.

Índice general

Índice de Figuras **XI**

Índice de Tablas **XIII**

Índice de Listings **XV**

1. Introducción **1**

1.1. Motivación del proyecto 1

1.2. Objetivos y enfoque 1

1.3. Organización de la memoria 2

2. Librería pygrib **3**

2.1. Ficheros GRIB 3

2.1.1. GRIB_API 4

2.2. Librería pygrib 5

2.2.1. Descarga e instalación 5

2.2.2. Clases, variables y métodos 6

2.3. Ejemplos de uso 11

2.3.1. Ejemplo de obtención de datos 11

2.3.2. Ejemplo de tratamiento de datos 12

2.4. Librerías similares 13

3. Aprendizaje automático en Python **15**

3.1. Aprendizaje automático 15

3.2. Modelos lineales 16

3.2.1. Mínimos cuadrados ordinarios 16

3.2.2. Regresión Ridge 17

3.3. Librería sklearn 17

3.3.1. Ejemplo de uso de sklearn 18

4. Diseño y desarrollo **21**
4.1. Herramientas software . . . 21
4.1.1. Librería NumPy . . . 21
4.1.2. Librería pandas . . . 22
4.1.3. Librería time . . . 24
4.1.4. Herramienta cron . . . 24
4.1.5. Bases de datos . . . 25
4.1.6. Documentación del código . . . 25
4.1.7. Control de versiones . . . 25
4.2. Diseño . . . 26
4.3. Desarrollo . . . 27
4.3.1. Manejo de datos meteorológicos . . . 28
4.3.2. Manejo de producciones energéticas . . . 31
4.3.3. Creación del modelo predictivo . . . 31
4.4. Experimentos realizados . . . 31

5. Conclusiones **35**

Glosario de acrónimos **37**

Bibliografía **39**

A. Librería netCDF4 **41**
A.1. Ficheros netCDF . . . 41
A.2. Librería netCDF4 . . . 42
A.2.1. Descarga e instalación . . . 43
A.2.2. Clases, variables y métodos . . . 43
A.3. Ejemplos de uso . . . 46
A.3.1. Ejemplo de creación de ficheros netCDF . . . 46
A.3.2. Ejemplo de obtención de datos de ficheros netCDF . . . 46
A.3.3. Ejemplo de conversión de netCDF a Dataframe . . . 46

B. Sphinx y Read The Docs **51**
B.1. Sphinx . . . 51
B.1.1. Instalación y creación de un proyecto . . . 51
B.1.2. Generación automática de documentación . . . 52
B.2. Read The Docs . . . 53
B.2.1. Uso de Read The Docs . . . 53
B.2.2. Read The Docs y autodoc . . . 54

C. Script de comparación de formatos **55**

Índice de Figuras

3.1. Mínimos cuadrados vs regresión Ridge 19

4.1. Diagrama del proceso de datos 26

4.2. Comparativa de alphas 32

Índice de Tablas

2.1. Códigos de niveles de presión 8

4.1. Ejemplo `Series` 22
4.2. Ejemplo `Dataframe` 22
4.3. Ejemplo parcial de un Dataframe histórico 27
4.4. Comparación de formatos para ficheros intermedios 28
4.5. Comparación de tamaños para Dataframes 29
4.6. Comparación de tiempos para Dataframe 29

Índice de Listings

2.1. Ejemplo de sección 0 de un mensaje GRIB 4
2.2. Ejemplo de sección 1 de un mensaje GRIB 5
2.3. Ejemplo de sección 2 de un mensaje GRIB 6
2.4. Ejemplo de sección 4 de un mensaje GRIB 7
2.5. Ejemplo de sección 5 de un mensaje GRIB 7
2.6. Obtención de datos del fichero 12
2.7. Obtención de datos de un mensaje 12
2.8. Filtrado de datos del fichero 13
2.9. Ejemplo de tratamiento de ficheros grib 14
3.1. Función para comparar modelos graficamente 19
3.2. Confección de los datos a usar en los modelos 20
3.3. Mínimos cuadrados en sklearn 20
3.4. Regresión Ridge en sklearn 20
3.5. Creación de la gráfica comparativa entre mínimos cuadrados y Ridge 20
4.1. Creación de la matriz de datos 33
4.2. Obtención del valor óptimo de alpha 34
4.3. Creación del modelo ridge óptimo 34
A.1. Fichero netCDF descomprimido 42
A.2. Ejemplo de creación de ficheros netCDF 47
A.3. Ejemplo de obtención de datos de ficheros netCDF 48
A.4. Ejemplo de conversión de netCDF a Dataframe 49
C.1. Script de comparación de tiempos 55

1 Introducción

1.1. Motivación del proyecto

De un tiempo a esta parte, las energías renovables como la solar o la eólica son más importantes y tienen más presencia en nuestras vidas. El principal problema de este tipo de energías es que dependen de fuentes muy irregulares por lo que las compañías eléctricas no las pueden usar como única fuente de producción, sino como soporte a la generación energética mediante fuentes tradicionales.

Por esta razón, para las compañías es muy importante saber cuánta energía van a obtener a partir de las renovables ya que así pueden ajustar más cuánto pueden vender.

Teniendo esto en cuenta, el objetivo de este trabajo ha sido el de estudiar las herramientas disponibles en Python para la gestión de predicciones meteorológicas y su explotación mediante la aplicación de aprendizaje automático en un contexto de energías renovables, específicamente, la energía eólica.

Este trabajo se ha realizado en el departamento de salud y energía del IIC (Instituto de energía del conocimiento) y se han utilizado datos sobre la producción de energía eólica de la península y Baleares desde 2015 hasta 2017, ambos incluidos.

1.2. Objetivos y enfoque

Como ya se ha comentado, el objetivo de este trabajo de fin de grado ha sido el estudio de los diferentes métodos de gestión de predicciones meteorológicas y de la aplicación de aprendizaje automático en Python sobre dichas predicciones para predecir la producción energética que generan las energías renovables, más en concreto la energía eólica. Para ello se ha simulado un sistema de recepción, tratamiento y procesado de datos lo más fiel a la realidad posible.

Para crear este sistema, primero hemos estudiado varios tipos de ficheros utilizados para guardar datos meteorológicos y cómo extraer la información contenida en ellos mediante las librerías de Python. Una vez extraídos estos datos, hemos investigado sobre la mejor forma de tratarlos y guardarlos para que la creación de la matriz de datos a usar en el modelo de predicción sea lo más óptima posible. Finalmente se han estudiado los fundamentos básicos del

aprendizaje automático y como aplicarlos en Python mediante la librería `sklearn`, consiguiendo así las predicciones que buscábamos.

1.3. Organización de la memoria

Esta memoria consta de 5 capítulos:

- Capítulo 1: motivación y objetivos del trabajo.
- Capítulo 2: introducción a los ficheros GRIB y la librería `pygrib`.
- Capítulo 3: introducción al aprendizaje automático y la librería `sklearn`.
- Capítulo 4: descripción de las herramientas utilizadas y pasos y decisiones tomadas a lo largo de la realización del trabajo.
- Capítulo 5: visión general del trabajo.

Consta también de tres anexos:

- Anexo A: introducción a los ficheros netCDF y la librería `netCDF4`.
- Anexo B: breve explicación de cómo se ha documentado y mantenido al día el código realizado.
- Anexo C: script realizado para comparar formatos de guardado de ficheros.

2

Librería pygrib

2.1. Ficheros GRIB

GRIB (GRIdded Binary) es un formato creado por la Organización Meteorológica Mundial (conocida como WMO por sus siglas en inglés) en 1985 para mejorar los formatos equivalentes que ya había, como GRID o GRAF [1]. En comparación con estos, los ficheros GRIB son más auto-explicativos y cuentan con un sistema de compresión propio. Este tipo de ficheros son altamente utilizados en el campo de la meteorología para guardar datos sobre las predicciones y los históricos de datos.

Los ficheros GRIB constan de una serie de mensajes o registros que contienen la información meteorológica de un punto específico. Estos mensajes tienen 6 secciones [1]:

0. Sección del indicador: en esta sección se encuentra la longitud total del mensaje y el número de edición de GRIB. La última edición publicada es la 2.

1. Sección de definición de producto: en esta sección podemos encontrar información genérica sobre el mensaje, como el centro en el que se ha creado la información o la fecha en la que se tomaron los datos. En esta sección cabe destacar los indicadores a partir de los que podemos obtener el nombre de la variable incluida en el mensaje, el nivel al que se han tomado los datos o el horizonte de la toma de datos.

2. Sección de descripción de la matriz de datos: en esta sección se encuentra la información relacionada con la matriz de datos, como el número de filas y columnas o el tipo de datos representados. Esta sección es opcional, aunque altamente recomendable.

3. Sección del mapa de bits: esta sección incluye un mapa de bits predefinido por el centro meteorológico. Consiste en una serie de bits que corresponden a las diferentes coordenadas de la matriz de datos. Un 1 en esta cadena de bits implica que hay datos para esas coordenadas y un 0 la ausencia de ellos. Esta sección es opcional.

4. Sección de datos binarios: en esta sección se encuentras los datos empaquetados y la información necesaria para desempaquetarlos.

```
#==============   MESSAGE 1 ( length=1638 )               ==============
1-4       identifier = GRIB
5-7       totalLength = 1638
8         editionNumber = 1
```

Listing 2.1: Ejemplo de sección 0 de un mensaje GRIB

5. Sección final: en esta sección está el código ASCII *'7777'*. Este valor es arbitrario y está definido por defecto. Sirve para indicar de forma legible para los humanos que el mensaje ha terminado y al ordenador que a partir de este punto ya no hay más datos.

Estos mensajes, al imprimirlos por pantalla utilizando el comando

`grib_list test_20170301.grib -s`

que viene incluido al instalar la GRIB_API, tienen el siguiente formato

`144:100 metre V wind component:m s**-1 (instant):regular_ll:surface:level 0:fcst time 60 hrs :from 201703010000.`

De aquí podemos obtener una información preliminar sobre el mensaje, como el número de mensaje (144), el tipo de variable que contiene (100 metre V wind component), cuándo se tomó la medida (01/03/2017 a las 00:00), etc. Para ver toda la información de los mensajes podemos utilizar el comando

`grib_dump -O gribfile.grib`

que viene incluido al instalar la GRIB_API. Este comando nos ofrece de forma legible la información de las diferentes secciones de un mensaje, como podemos ver en los listings 2.1, 2.2, 2.3, 2.4 y 2.5, que contienen la información de las secciones 0, 1, 2, 4 y 5 respectivamente.

A toda la información contenida en los mensajes podemos acceder mediante el uso de librerías específicas para ciertos lenguajes de programación. En Python esta librería se llama `pygrib` y nos permite tratar los ficheros GRIB y los mensajes contenidos en ellos como objetos.

2.1.1. GRIB_API

El European Centre for Medium-Range Weather Forecasts (conocido como ECMWF) dispone de una herramienta gratuita llamada GRIB_API que permite trabajar con ficheros GRIB (edición 1 y edición 2) desde C, FORTRAN y Python. Esta herramienta nos ofrece tanto métodos y funciones para usar a la hora de programar como comandos que nos permiten tratar fácilmente con ficheros GRIB desde la línea de comandos de linux.

Para instalarla en linux, se puede hacer mediante la herramienta `CMake` (como podemos ver en la página web de la ECMWF) o con los comandos

`sudo apt-get install libgrib-api-dev` o `conda install -c eumetsat ecmwf-grib-api`.

También hay una versión para Windows, pero está en estado experimental y no aseguran su mantenimiento [2].

A partir de finales de 2018, esta herramienta quedará obsoleta y será sustituida por ecCodes, con la cuál se pretende unificar la GRIB_API y la aplicación BUFRDC (para trabajar con ficheros BUFR) en una sola. EcCodes se puede utilizar desde C, Python y FORTRAN 90 (FORTRAN 77 ya no está soportado) y dispone de los mismos métodos y funciones que la GRIB_API manteniendo lo mas fielmente posible sus opciones y funcionamiento [3]. La migración de una aplicación a la otra es relativamente sencilla, ya que las herramientas para línea de comandos mantienen los nombres y las definiciones de funciones conservan la funcionalidad, los argumentos y los retornos. Lo único que habría que modificar es el nombre de las funciones en nuestro

```
======================   SECTION_1 ( length=52, padding=0 )    ======================
1-3       section1Length = 52
4         table2Version = 128
5         centre = 98 [European Centre for Medium-Range Weather Forecasts (grib1/0.table) ]
6         generatingProcessIdentifier = 148
7         gridDefinition = 255
8         section1Flags = 128 [10000000]
9         indicatorOfParameter = 165 [10 metre U wind component  (m s**-1)  (grib1/2.98.128.
    table) ]
10        indicatorOfTypeOfLevel = 1 [Surface  (of the Earth, which includes sea surface)  (
    grib1/local/ecmf/3.table , grib1/3.table) ]
11-12     level = 0
13        yearOfCentury = 17
14        month = 9
15        day = 6
16        hour = 0
17        minute = 0
18        unitOfTimeRange = 1 [Hour (grib1/4.table) ]
19        P1 = 0
20        P2 = 0
21        timeRangeIndicator = 1 [Initialized analysis product for reference time  (P1=0)  (
    grib1/local/ecmf/5.table , grib1/5.table) ]
22-23     numberIncludedInAverage = 0
24        numberMissingFromAveragesOrAccumulations = 0
25        centuryOfReferenceTimeOfData = 21
26        subCentre = 0 [Unknown code table entry (grib1/0.ecmf.table) ]
27-28     decimalScaleFactor = 0
29-40     reservedNeedNotBePresent = 12 {
                00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
             } # pad reservedNeedNotBePresent
41        localDefinitionNumber = 1 [MARS labelling or ensemble forecast data (grib1/
    localDefinitionNumber.98.table) ]
42        marsClass = 1 [Operational archive (mars/class.table) ]
43        marsType = 9 [Forecast (mars/type.table) ]
44-45     marsStream = 1025 [Atmospheric model (mars/stream.table) ]
46-49     experimentVersionNumber = 0001
50        perturbationNumber = 0
51        numberOfForecastsInEnsemble = 0
52        padding_local1_1 = 1 {
                  00
                } # pad padding_local1_1
```

Listing 2.2: Ejemplo de sección 1 de un mensaje GRIB

código, que en la mayoría de los casos pasa de tener el formato `grib_nombre_de_la_funcion()` a tener el formato `codes_nombre_de_la_funcion()`.

2.2. Librería pygrib

Pygrib es la librería Python que permite leer y escribir ficheros GRIB. Esta librería es una interfaz de la GRIB_API de la librería de C del ECWMF (European Centre for Medium-Range Weather Forecasts) y la librería grib2 de C del NCEP (National Centers for Environmental Prediction).

2.2.1. Descarga e instalación

Para utilizar esta librería, se recomienda usar entornos UNIX, ya que, aunque posible, la instalación de la misma en Windows es más complicada. La descarga de la misma desde Linux puede hacerse de varias formas.

- Mediante el comando `pip`: desde la consola de comandos es suficiente con usar el comando `pip install pygrib`. El comando `pip` se instala automáticamente al instalar Python.

```
======================   SECTION_2 ( length=32, padding=0 )    ======================
1-3       section2Length = 32
4         numberOfVerticalCoordinateValues = 0
5         pvlLocation = 255
6         dataRepresentationType = 0 [Latitude/Longitude Grid (grib1/6.table) ]
7-8       Ni = 45
9-10      Nj = 17
11-13     latitudeOfFirstGridPoint = 29500
14-16     longitudeOfFirstGridPoint = 341500
17        resolutionAndComponentFlags = 128 [10000000]
18-20     latitudeOfLastGridPoint = 27500
21-23     longitudeOfLastGridPoint = 347000
24-25     iDirectionIncrement = 125
26-27     jDirectionIncrement = 125
28        scanningMode = 0 [00000000]
29-32     zero =
```

Listing 2.3: Ejemplo de sección 2 de un mensaje GRIB

- Con la plataforma anaconda: se trata de una distribución gratuita de Python que incluye el gestor de paquetes conda. Con este podemos utilizar el comando
  `conda install -c conda-forge pygrib`
  para instalar la librería.

- Desde github: para instalar la librería hay que realizar las siguientes acciones:

  1. Comprobar que cumplimos los siguientes requisitos:
     - Tenemos instalada la distribución de Python 2.4 o una versión superior.
     - Tenemos instalada la librería `numpy` en la versión 1.2.1 o superior.
     - Tenemos instalada la librería `pyproj` o la librería `matplotlib` y las herramientas de `basemap`.
     - Tenemos instalada la librería de C GRIB_API. Para un completo funcionamiento de esta librería, necesitaremos tener las librerías Jasper o OpenJPEG, librerías encargadas de trabajar con el formato de compresión JPEG2000 (Joint Photographic Experts Group 2000), y la librería PNG, encargada de trabajar con el formato de compresión PNG (Portable Network Graphics).
  2. Descargar los ficheros desde el repositorio de GitHub o clonar el mismo.
  3. Modificar el nombre del fichero `setup.cfg.template` por `setup.cfg` y seguir las instrucciones del mismo.
  4. Ejecutar el comando `python setup.py build`.
  5. Ejecutar el comando `python setup.py install`, con permisos de superusuario si fuese necesario.
  6. Ejecutar el comando `python test.py` para comprobar que se ha instalado correctamente.

  Además del código, en el repositorio de github podemos encontrar ejemplos de uso y documentación sobre la librería.

2.2.2. Clases, variables y métodos

En Python los objetos GRIB constan de tres subclases:

- `open`: permite tratarlos como cualquier otro fichero.

```
======================   SECTION_4 ( length=1542, padding=0 )   ======================
1-3       section4Length = 1542
4         dataFlag = 8 [00001000]
5-6       binaryScaleFactor = -11
7-10      referenceValue = -11.1856
11        bitsPerValue = 16
12-1542   values = (765,1531) {
-8.1714601517e+00, -7.8574953079e+00, -7.8701906204e+00, -8.0557374954e+00, -7.8013429642e+00,
     -7.2744874954e+00, -7.0703859329e+00, -7.0640382767e+00,
-6.8081789017e+00, -6.3360109329e+00, -6.0459718704e+00, -5.7173585892e+00, -5.3775148392e+00,
     -5.4229249954e+00, -5.5030031204e+00, -5.1489992142e+00,
-4.9307374954e+00, -4.9229249954e+00, -4.9805421829e+00, -5.1865968704e+00, -5.1665773392e+00,
     -4.9907960892e+00, -4.8438234329e+00, -4.5528078079e+00,
-4.4165773392e+00, -4.5342531204e+00, -4.5366945267e+00, -4.5274171829e+00, -4.4072999954e+00,
     -4.1651124954e+00, -4.2041749954e+00, -4.3374757767e+00,
-4.3863039017e+00, -4.3872804642e+00, -4.2451906204e+00, -4.0020265579e+00, -3.9336671829e+00,
     -3.9492921829e+00, -3.6270265579e+00, -3.2334718704e+00,
-3.2354249954e+00, -3.0542726517e+00, -2.7891359329e+00, -2.7583742142e+00, -2.5777101517e+00,
     -8.0786867142e+00, -8.0962648392e+00, -7.9746828079e+00,
-7.4956789017e+00, -7.2251710892e+00, -7.2481203079e+00, -7.0840578079e+00, -6.5825929642e+00,
     -6.0708742142e+00, -5.9995851517e+00, -6.2993898392e+00,
-6.1865968704e+00, -5.6656007767e+00, -5.3545656204e+00, -5.2349367142e+00, -5.0689210892e+00,
     -4.8643312454e+00, -4.7154054642e+00, -4.8169679642e+00,
-5.0772218704e+00, -5.2124757767e+00, -5.2759523392e+00, -5.2432374954e+00, -5.1763429642e+00,
     -5.2681398392e+00, -5.1934328079e+00, -4.8482179642e+00,
-4.6314210892e+00, -4.4580812454e+00, -4.3203859329e+00, -4.2710695267e+00, -4.2295656204e+00,
     -4.2798585892e+00, -4.2608156204e+00, -4.1343507767e+00,
-4.0796632767e+00, -3.9512453079e+00, -3.6411867142e+00, -3.3335695267e+00, -3.2490968704e+00,
     -3.1240968704e+00, -2.6870851517e+00, -2.5957765579e+00,
-2.5899171829e+00, -2.2652101517e+00, -8.1392335892e+00, -8.3545656204e+00, -8.3228273392e+00,
     -8.0698976517e+00, -8.0415773392e+00, -7.5108156204e+00,
-6.5645265579e+00, -6.1382570267e+00, -6.2183351517e+00, -6.1548585892e+00
... 665 more values
} # data_g1simple_packing values
```

Listing 2.4: Ejemplo de sección 4 de un mensaje GRIB

```
======================   SECTION_5 ( length=4, padding=0 )     ======================
1-4       7777 = 7777
```

Listing 2.5: Ejemplo de sección 5 de un mensaje GRIB

- `gribmessage`: permite obtener la información de los diferentes mensajes por los que está compuesto.
- `index`: permite crear índices propios para organizar los mensajes.

Subclase `open`

Esta clase nos permite tratar los ficheros GRIB como cualquier otro fichero, por lo que se pueden utilizar las funciones `open`, `seek`, `tell`, `read`, `readline` y `close`. Estos métodos funcionan igual que con ficheros comunes, a excepción de que en lugar de trabajar en bytes lo hacen en mensajes, es decir, si en un fichero utilizamos `read(10)` para leer los siguientes 10 bytes del fichero, en un fichero GRIB esta misma llamada nos devolvería los siguientes 10 mensajes.

Además de esto, esta subclase aporta las siguientes variables:

- `messagenumber`: número de mensaje al cual apunta actualmente el iterador.
- `messages`: número total de mensajes que contiene el fichero.
- `name`: nombre del fichero GRIB abierto actualmente.

Value	Meaning
00	Reserved
01	Surface (of the Earth, which includes sea surface)
02	Cloud base level
03	Cloud top level
04	0 deg (C) isotherm level
05	Adiabatic condensation level(parcel lifted from surface)
06	Maximum wind speed level
07	Tropopause level
08	Nominal top of atmosphere
09	Sea bottom
10-99	Reserved
100	Isobaric level
101	Layer between two isobaric levels
102	Mean sea level
103	Fixed height level
104	Layer between two height levels above mean sea level
105	Fixed height above ground
106	Layer between two height levels above ground
107	Sigma level
108	Layer between two sigma levels
109	Hybrid level
110	Layer between two hybrid levels
111	Depth below land surface
112	Layer between two depths below land surface
113	Isentropic (theta) level
114	Layer between two isentropic levels
121	Layer between two isobaric surfaces (high precision)
125	Height level above ground (high precision)
128	Layer between two sigma levels (high precision)
141	Layer between two isobaric surfaces (mixed precision)
160	Depth below sea level
200	Entire atmosphere considered as a single layer
201	Entire ocean considered as a single layer

Cuadro 2.1: Códigos de niveles de presión

También están disponibles los siguientes métodos:

- `message(N)`: devuelve el n-ésimo mensaje del fichero.
- `select(**kwargs)`: este método devuelve una lista con los mensajes que satisfagan las condiciones pasadas como argumentos. Estas condiciones se pueden satisfacer de tres maneras:
  - Comprobando que la clave seleccionada sea igual al valor indicado. Ejemplo:
    `Select(name='100 metre U wind component')`
  - Comprobando que la clave es igual a alguna de los valores incluidos en una lista. Ejemplo:
    `Select(name=['100 metre U wind component', 'Surface pressure'])`
  - Comprobando que la clave cumpla la condición de un objeto con el método `__call__` (que permite que un objeto sea llamado como si fuese una función), es decir, sea igual al retorno de algún método o función lambda. Ejemplo:
    `Select(level= lambda l: l>0)`

Subclase `gribmessage`

Cada uno de los mensajes contenidos en el fichero GRIB consta de una serie de variables y métodos, a los que podemos acceder como en cualquier otro objeto de Python, y de una lista de parámetros organizados como si de un diccionario se tratase, que, dispuestos como en las secciones descritas anteriormente, aportan toda la información de los datos meteorológicos guardados. En el caso de los ficheros del ECMWF cada mensaje tiene 178 de estos parámetros. Algunos de estos son:

- `GRIBEditionNumber`: edición de GRIB utilizada para crear el fichero.
- `totalLength`: tamaño del mensaje.
- `centreDescription`: nombre del centro en el que se han realizado las predicciones.
- `level`: nivel de presión al que se han tomado las medidas. Puede tomar valores de 0 a 201, siendo cada nivel una altura y/o región específica, como podemos ver en la tabla 2.1 [1].
- `yearOfCentury, month, day, hour, minute, second`: fecha en la que se ha realizado la medida.
- `name`: nombre de la medida tomada.
- `Ni, Nj`: número de filas y columnas que tiene la matriz de datos.
- `numberOfDataPoints, numberOfValues, numberOfCodedValues, getNumberOfValues`: número de valores que hay guardados. Si todas las medidas se han tomado correctamente los cuatro valores deben ser iguales.
- `latLonValues`: lista con los valores de las latitudes, las longitudes y los valores registrados para las mismas. Esta lista tiene el formato `[latitud 0, longitud 0, valor 0, latitud 1, longitud 1, valor 1 ....]`.
- `latitudes, longitudes`: listas con los valores de las latitudes y las longitudes respectivamente.
- `distinctLatitudes, distinctLongitudes`: listas con los diferentes valores de latitudes y longitudes que nos podemos encontrar.

- `missingValue`: valor que representa que falta un valor registrado.
- `values`: matriz con los valores registrados. En esta matriz, las filas están representadas por las latitudes y las columnas por las longitudes.
- `maximum, minimum, average`: valor máximo, mínimo y la media de los valores registrados.
- `numberOfMissing`: número de valores de la matriz de datos que no se han registrado correctamente.
- `standardDeviation, kurtosis, skewness`: valores estadísticos sobre la matriz de datos: desviación típica, curtosis y asimetría estadística.
- `analDate`: fecha en la que se realizó la predicción meteorológica.
- `validDate`: fecha válida de la predicción meteorológica. Si esta se ha hecho durante un rango de tiempo, esta fecha será la última de dicho rango.

Para acceder a estos campos, lo debemos hacer como si se tratase de un diccionario. Por ejemplo si nuestro objeto se llama `msg`, para acceder a los valores o a la fecha de análisis lo haríamos como `msg['values']` o `msg['analDate']` respectivamente.

Además de estos campos, esta subclase contiene una serie de variables y métodos que nos facilitan la obtención de ciertos datos.

- Variables:
  - `messagenumber`: número del mensaje dentro del fichero GRIB.
  - `analDate`: fecha en la que se realizó la predicción meteorológica.
  - `validDate`: fecha válida de la predicción meteorológica. Si esta se ha hecho durante un rango de tiempo, esta fecha será la última de dicho rango.
- Métodos:
  - `keys()`: devuelve una lista con todos los nombres de los parámetros que tiene el mensaje.
  - `latlons()`: devuelve dos arrays con los valores de las latitudes y las longitudes en grados.
  - `tostring()`: devuelve el mensaje como cadena binaria.
  - `data(lat1=None, lat2=None, lon1=None, lon2=None)`: devuelve los valores, las longitudes y las latitudes de una región definida por los valores pasados como argumento en formato de grados decimales.
  - `has_key(key)`: comprueba si el mensaje tiene el parámetro dado.
  - `is_missing(key)`: devuelve `True` si el parámetro pasado como argumento es inválido o el valor asociado es el valor por defecto de valor desconocido (dado por el parámetro `missingValue`. Si en no se ha indicado lo contrario este valor es `9999`).

Para acceder a las variables y los métodos, lo debemos hacer como con cualquier otro objeto de Python. Por ejemplo si nuestro objeto se llama `msg`, para obtener las latitudes y longitudes o a la fecha de análisis lo haríamos como `msg.latlons()` o `msg.analDate` respectivamente.

Subclase `index`

Esta clase nos permite abrir los ficheros GRIB indexados por las claves pasadas como argumento en su constructor. Esta clase, al contrario que la clase `open` no los abre como ficheros normales, por lo que no contamos con los métodos propios de manejo de ficheros de Python ni se puede iterar sobre los mensajes. A cambio, el uso de esta clase nos aporta velocidad a la hora de filtrar mensajes.

Esta clase contiene las siguientes variables:

- `keys`: lista con las claves que actúan como índice del fichero.
- `name`: nombre del fichero
- `types`: si las variables usadas como índice tienen un tipo de datos definido, devuelve una lista con dichos tipos.

También dispone de los siguientes métodos:

- `close()`: permite cerrar el fichero
- `write(filename)`: guarda el índice en un fichero de texto. Este método no guarda ni el nombre de las claves utilizadas como índice ni su tipo de dato en caso de tenerlo.
- `select(**kwargs)`: nos permite filtrar los mensajes del fichero usando las variables que forman el índice. Para que funcione hay que asignarles un valor a todos los índices. Este valor tiene que ser único, es decir, no puede ser ni una lista de valores ni un objeto con el método `__call__` (que permite que un objeto sea llamado como si fuese una función). Ejemplo:

```
import pygrib

grib = pygrib.index('fichero.grib','level','name')

selection = grib.select(level=0,name='10 metre U wind component')

for grb in selection:
    print(grb)
```

  También se pueden filtrar mensajes mediante el uso del método`__call__` de la propia clase. La llamada `grib.select(level=0)` es equivalente a la llamada `grib(level=0)`. Ejemplo:

```
import pygrib

grib = pygrib.index('fichero.grib','level','name')

selection = grib(level=0,name='10 metre U wind component')

for grb in selection:
    print(grb)
```

No es recomendable el uso de esta clase si el fichero GRIB a tratar contiene campos con valores múltiples.

2.3. Ejemplos de uso

2.3.1. Ejemplo de obtención de datos

En el siguiente ejemplo, podemos ver varios casos de obtención de datos de un fichero GRIB. El código lo podemos dividir en tres partes:

```
import pygrib

grib = pygrib.open('./Historicoprueba2.grib')

# Imprime todas las claves de un mensaje
print('Keys:\n',grib[1].keys())

# Imprime todos los mensajes del fichero
print('\nMensajes:')
for grb in grib:
    print(grb)
```

Listing 2.6: Obtención de datos del fichero

```
# Imprime los campos del primer mensaje y su contenido
print('\nContenido del primer mensaje')
for i,key in enumerate(grib[1].keys()):
    if key == 'analDate':
        print(i,key,grib[1].analDate.strftime(' %Y %m %d %H'))
    elif key == 'validDate':
        print(i, key, grib[1].validDate.strftime(' %Y %m %d %H'))
    else:
        print(i,key,grib[1][key])

print('\nNumero de mensajes en el fichero\n',grib.messages)

m1 = grib.message(1) # Obtención del primer mensaje
v1 = m1.values # Obtención de la matriz de valores
l1,l2 = m1.latlons() # Obtención de las matrices de latitudes y longitudes

print('\nMensaje:\n',m1)
print('\nTamaño de la matriz de valores\n', v1.shape)
print('\nTamaño de la matriz de latitudes\n',l1.shape)
print('\nTamaño de la matriz de longitudes\n',l2.shape)
print('\nNumero de mensaje dentro del fichero\n',m1.messagenumber)
print('\nSubconjunto de datos del mensaje\n',m1.data(lat1=52,lon1=16,lat2=53,lon2=17))
```

Listing 2.7: Obtención de datos de un mensaje

- Obtención de datos genéricos del fichero: aquí abrimos el fichero, y a partir de ahí vamos obteniendo tanto los mensajes que lo componen como los diferentes valores de cada uno de ellos. Un ejemplo se puede ver en el listing 2.6.
- Obtención de datos de un mensaje específico: aquí accedemos a uno de los mensajes, podemos hacerlo como si fuese una lista (`grib[1]`) o mediante un método (`grib.message(1)`); al acceder a los mensajes de estas formas, el primer mensaje es el 1, no el 0, como en las listas típicas de Python. Además de esto podemos ver el uso del método `data()` que nos permite obtener un subconjunto de datos, en este caso los correspondientes a la zona geográfica de las coordenadas (52,16) a las coordenadas (53,17). Un ejemplo se puede ver en el listing 2.7.
- Filtrado de mensajes: También podemos ver los tres modos de búsqueda para el método `select()` que ofrece la subclase open (sección 2.2.2). Un ejemplo se puede ver en el listing 2.8.

2.3.2. Ejemplo de tratamiento de datos

En este ejemplo (listing 2.9), vamos obteniendo los datos del fichero grib para ir metiéndolos en un diccionario, el cual utilizamos para volcar fácilmente todos los valores en varios `dataframes`, uno por cada fecha de toma de datos diferente. Estos `dataframes` tienen como índice la fecha en

```
print('\nMensajes pertenecientes a ''100 metre U wind component''\n')
messages = grib.select(name='100 metre U wind component')
for msg in messages:
    print(msg)

print('\nMensajes pertenecientes a ''100 metre U wind component'' y ''Surface pressure''\n')
messages = grib.select(name=['100 metre U wind component', 'Surface pressure'])
for msg in messages:
    print(msg)

print('\nMensajes con una media de valores entre 200 y 300\n')
messages = grib.select(average= lambda n: n>=200 and n<=300)
for msg in messages:
    print(msg)

grib.close()
```

Listing 2.8: Filtrado de datos del fichero

la que se han tomado las medidas y como columnas el nombre de la variable y las coordenadas de los datos. La creación y manejo de los `dataframes` lo hacemos con la librería `pandas` (sección 4.1.2).

2.4. Librerías similares

Para el tratamiento de datos meteorológicos, existen otras librerías que funcionan de manera similar. Un ejemplo de esto es la librería netCDF4, que nos permite trabajar con ficheros netCDF desde Python. Este tipo de fichero fue creado por Unidata [4] y se caracteriza por ser autodescriptivo, multiplataforma y por permitir guardar los datos en forma de array de una forma clara y sencilla.

Una descripción mas detallada de este tipo de ficheros y del funcionamiento de la librería netCDF4 la podemos encontrar en el anexo A

```
import pandas
import pygrib

def grib_to_df(grib_file=None, output=None, days=None):
    """
    Introduce los datos de un fichero grib en un dataframe

    Los datos de la medición se introducen en un dataframe con la fecha
    de la predicción como índice.

    :param grib_file: nombre del fichero del que tomar los datos
    :param output: directorio en el que guardar el dataframe creado
    :param days: número de días a incluir en el diccionario
                            None -> todos los días (valor por defecto)
                            0 -> Día de la toma de datos
                            1 -> Día de la toma de datos más un día
    :return: lista con los dataframes creados    """

    grib = pygrib.open(grib_file)

    dates = []
    for grb in grib:
        if not grb.analDate in dates:  # obtención de las fechas de toma de datos
            dates.append(grb.analDate)

    dicc = {}
    for date in dates:
        # comprobación del rango de datos a incluir en el diccionario
        if days != None:
            time_max = date + datetime.timedelta(days=days)

        dicc[date] = {}
        data = grib.select(analDate=date)

        for grb in data:

            # Comprobamos que no nos salimos del rango de fechas a incluir
            time = date + datetime.timedelta(hours=grb['P1'])
            if days != None and time.day > time_max.day:
                continue

            values = grb.values
            lat, lon = grb.latlons()
            variable_name = grb['name']

            if not time in dicc[date]:
                dicc[date][time] = {}

            # obtenemos las coordenadas y valores de las mismas
            # y las introducimos ene l diccionaro
            for i in range(lat.shape[0]):
                for j in range(lat.shape[1]):
                    coords = lat[i][j], lon[i][j]
                    key = variable_name + str(coords)
                    dicc[date][time][key] = values[i][j]

    dataframes = []
    for date in dicc:
        df = pd.DataFrame.from_dict(dicc[date], orient='index')
        df.index.name = 'prediction date'
        df.insert(0, 'generation date', date)

        fecha = date.strftime('%Y%m%d')
        df.to_pickle(output + grib.name[-25:-8] + fecha + '_pickle')
        dataframes.append(df)

    grib.close()

    return dataframes
```

Listing 2.9: Ejemplo de tratamiento de ficheros grib

3

Aprendizaje automático en Python

En este capítulo se encuentra una breve introducción al aprendizaje automático, viendo qué tipos hay y explicando los dos modelos que se han estudiado, mínimos cuadrados ordinarios y regresión Ridge. Finalmente se explica el funcionamiento básico de la librería Python `sklearn`, que nos permite implementar diferentes modelos predictivos.

3.1. Aprendizaje automático

El aprendizaje automático es la rama de la Inteligencia Artificial que en cierto sentido permite a las máquinas aprender para poder hacer predicciones sobre un conjunto de datos. Este campo de la informática está muy relacionado con la computación estadística y la minería de datos, ya que la metodología de aprendizaje más generalizada es la utilización de grandes cantidades de datos para encontrar las diferentes formas de clasificarlos o de construir modelos.

En función de cómo estén dispuestos estos datos, podemos distinguir dos grandes ramas, el aprendizaje supervisado y el aprendizaje no supervisado.

Aprendizaje supervisado

Este tipo de aprendizaje se caracteriza por tener los datos divididos en dos grupos, los datos que representan las distintas entradas posibles y los que representan las salidas correspondientes. Este conjunto de datos, se divide a su vez en dos partes, el conjunto de entrenamiento o *training set* y el conjunto de pruebas o *testing set*. El *training set* lo utilizamos para alimentar al algoritmo y que este cree una función que relacione lo más precisamente posible las entradas con las salidas esperadas. El *testing set* lo utilizamos para comprobar la eficacia del modelo creado.

Dentro de este tipo de aprendizaje podemos encontrar dos tipos de problemas:

- Problemas de clasificación: consisten en asignar a los datos una clase entre todas las posibles. Un ejemplo sería el de reconocer números escritos a mano en el que se tiene la imagen de una cifra y hay que asignarla a una de las diez clases posibles (de 0 a 9).

- Problemas de regresión: son problemas en los que la salida esperada consiste en una o más variables continuas. Un ejemplo sería el de determinar el precio de una vivienda en función de factores externos, como el vecindario, el año de construcción o el tamaño.

Aprendizaje no supervisado

En este tipo de aprendizaje los datos únicamente están compuestos por distintas entradas posibles; aquí no disponemos de las salidas esperadas como en el aprendizaje supervisado. Por esta razón, en este aprendizaje no tenemos forma de comprobar la eficacia del modelo con los datos iniciales.

El objetivo de este tipo de aprendizaje es la obtención de posibles relaciones y patrones existentes entre todos los datos del conjunto. Un ejemplo de aprendizaje no supervisado es la agrupación de los clientes de un supermercado en función de los productos que compran.

3.2. Modelos lineales

Un modelo lineal es un modelo matemático que pretende encontrar una relación de dependencia lineal entre una variable dependiente y y las variables independientes x_i dado un conjunto de datos $y_i, x_{i1}, ..., x_{ip}{}_{i=1}^{n}$. Para cada variable, esta relación la podemos expresar como

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p = \beta_0 + \sum_{i=1}^{p} \beta_i x_i,$$

donde y es la variable dependiente o resultado del modelo, x_i son las variables independientes o los datos introducidos en el modelo y β_i los parámetros que miden la influencia entre ambas variables. De estos parámetros, al β_0 se le conoce como ordenada.

Esta ecuación la podemos expresar en forma matricial como

$$\hat{Y} = X\beta$$

en la que tomamos que $X_0 = 1$.

El problema de la aplicación de modelos lineales al aprendizaje automático se basa en la obtención de los valores β_i. En función de cómo obtengamos dichos parámetros disponemos de varios métodos, como el de mínimos cuadrados ordinarios o el de la regresión de arista, más conocida como *Ridge Regression* en inglés.

3.2.1. Mínimos cuadrados ordinarios

Para hallar los valores de los parámetros, este método usa la diferencia entre el valor observado y el valor obtenido con el modelo, también llamado residuo. Este valor viene dado por la formula $\varepsilon = Y - X\beta$. La suma residual de cuadrados (RSS por sus siglas en inglés) ajusta estos valores en el modelo general y viene dada por la siguiente ecuación:

$$RSS(\beta) = \sum_{i=1}^{p} (y_i - x_i^T \beta)^2$$

Si definimos X como una matriz de $n \times p$ en la que cada fila es un vector con datos de la matriz de entrenamiento e Y es un vector de tamaño n con las salidas correspondientes podemos expresar la fórmula como

$$RSS(\beta) = (Y - X\beta)^T (Y - X\beta).$$

Si la matriz X^TX es invertible, entones al minimizar la suma residual de cuadrados, obtenemos que los parámetros tienen una solución única y esta viene dada por

$$\hat{\beta} = (X^TX)^{-1}X^Ty$$

3.2.2. Regresión Ridge

Al utilizar el método de los mínimos cuadrados ordinarios puede suceder que alguna de las variables que estamos utilizando para predecir sea linealmente dependiente de otra, lo que puede hacer que las predicciones no sean precisas o incluso no puedan calcularse los valores de los parámetros. Para solucionar esta situación podemos sesgar los valores de dichos parámetros penalizando los de mayor valor. Esto lo podemos hacer añadiendo un valor arbitrario a la función del cálculo de la suma residual de cuadrados, por lo que la expresión quedaría de la siguiente manera

$$RSS(\beta) = \sum_{i=1}^{p}(y_i - x_i^T\beta)^2 + \alpha\sum_{i=1}^{p}\beta_i^2$$

o en forma de matriz

$$RSS(\beta) = (y - X\beta)^T(y - X\beta) + \alpha\beta^T\beta$$

donde $\alpha > 0$ es el valor que limita el tamaño de los parámetros de regresión.

Si la matriz X^TX es invertible, entonces al minimizar la suma residual de cuadrados, obtenemos que los parámetros de regresión tienen una solución única y esta viene dada por

$$\hat{\beta} = (X^TX + \alpha I)^{-1}X^Ty$$

donde I es la matriz identidad.

3.3. Librería sklearn

Esta librería nos ofrece una amplia gama de herramientas para realizar tareas de aprendizaje automático. La podemos obtener mediante el comando

```
pip install -U scikit-learn
```

o con el comando

```
conda install scikit-learn
```

si disponemos de una distribución de Anaconda, aunque en este caso, la librería suele venir instalada por defecto.

Dentro de todos los módulos disponibles los más convenientes para empezar a utilizar esta librería son:

- `datasets`: contiene un conjunto de herramientas que nos permiten obtener datos que podemos usar fácilmente para entrenar y probar los diferentes modelos. Dispone de funciones tanto para cargar datos ya creados (por ejemplo podemos cargar datos para el análisis de los precios de casas con `datasets.load_boston()`) como para crearlos de manera automática y aleatoria, por ejemplo con la utilización del siguiente método:

  ```
  datasets.samples_generator.make_regression(n_samples=100, n_features=100, n_informative=10)
  ```

- `model_selection`: contiene un conjunto de herramientas que nos permiten trabajar con el conjunto total de los datos, específicamente para obtener estimadores o para dividir los datos en un set de entrenamiento y un set de pruebas, como por ejemplo:

  `model_selection.train_test_split(x,y,train_size=0.30).`

- `preprocessing`: contiene una serie de métodos que nos permiten preparar los datos para trabajar con ellos antes de crear el modelo, como por ejemplo `normalize()` que normaliza el conjunto de datos. Por defecto utiliza la norma l^2, es decir, para un vector x,

$$||x||_2 = \sqrt{\sum_{i=1}^{k} |x_k|^2}$$

- `linear_model`: contiene las implementaciones de diferentes modelos lineales como el de mínimos cuadrados ordinarios o el modelo Ridge. Todos estos modelos disponen de métodos para entrenar el modelo (`fit(traninig_data,target_values)`), para predecir valores (`predict(test_data)`) y para evaluar las predicciones (`score(test_data,test_target)`). Este tipo de evaluación se basa en el coeficiente de determinación, denominado R^2, que mide en qué porcentaje están relacionadas las predicciones realizadas y los datos reales. Esta medida oscila entre 0 y 1, siendo 0 lo peor (predicciones nada fiables) y 1 lo mejor (predicciones absolutamente precisas).

- `metrics`: contiene funciones que nos permiten evaluar los resultados obtenidos en las predicciones realizadas. Aquí nos podemos encontrar métodos como `r2_score()` que realiza la evaluación basada en el coeficiente de determinación o `mean_absolute_error()`, que nos indica el error medio cometido en las predicciones. Este valor es siempre positivo y cuanto más cercano a 0 mejor son las predicciones.

Además de estos, dispone de muchos otros módulos que nos permiten trabajar de diferentes maneras con los datos y aplicar distintos modelos. Una lista completa de los módulos disponibles la podemos encontrar en su página web.

3.3.1. Ejemplo de uso de sklearn

Un ejemplo de uso de esta librería, sería la creación de dos modelos para predecir sobre unos datos dados. Para realizar este ejemplo, primero hemos generado datos aleatoriamente, como podemos ver en el listing 3.2. En este caso creamos unos datos de 5000 muestras y los dividimos en 3500 muestras para entrenamiento y 1500 para pruebas.

Con estos datos, entrenamos y predecimos con un modelo de mínimos cuadrados y con una regresión Ridge, como podemos ver en los listings 3.3 y 3.4. En la regresión Ridge, calculamos α mediante validación cruzada, que consiste en crear subconjuntos independientes a partir de los datos iniciales en los que probamos varios valores de α con el fin de encontrar el óptimo.

Finalmente, como podemos ver en el listing 3.5, llamamos a la función creada en el listing 3.1, que mediante el método `learning_curve()` realiza automáticamente varios modelos con diferentes subconjuntos de los datos creados.

A partir de estos resultados, obtenemos una comparativa del rendimiento de ambos modelos como podemos ver en la figura 3.1. En ella podemos observar que para conjuntos con pocos elementos de entrenamiento, el método de mínimos cuadrados tiene menos tasa de acierto al predecir que con el método Ridge. En cambio, con conjuntos de muchos elementos de entrenamiento las tasas de acierto de ambos métodos se van acercando.

Figura 3.1: Mínimos cuadrados vs regresión Ridge

```
#!/usr/bin/env python

import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model,model_selection
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, title, X, y, color,ylim=None, cv=None, n_jobs=1,
    train_sizes=np.linspace(.1, 1, 20)):

    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv, n_jobs=
        n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,train_scores_mean +
        train_scores_std, alpha=0.1,color=color)
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,test_scores_mean +
        test_scores_std, alpha=0.1, color=color)
    plt.plot(train_sizes, train_scores_mean, '--', color=color,label="Training score " + title
        )
    plt.plot(train_sizes, test_scores_mean, '-', color=color,label="Tests score " + title)
    plt.legend(loc="best")
    return plt
```

Listing 3.1: Función para comparar modelos graficamente

```
# Creación de una muestra aleatoria de datos
x,y = datasets.samples_generator.make_regression(n_samples=5000, n_features=500, n_informative
    =400, n_targets=4, noise=500, random_state=7)

# División de los datos en conjunto de entrenamiento/conjunto de pruebas
x_train,x_test,y_train,y_test = model_selection.train_test_split(x,y,random_state=5,train_size
    =0.30)
```

Listing 3.2: Confección de los datos a usar en los modelos

```
# Mínimos cuadrados
lr = linear_model.LinearRegression()
lr.fit(x_train,y_train)
pred = lr.predict(x_test)
print(lr.score(x_test,y_test))
```

Listing 3.3: Mínimos cuadrados en sklearn

```
# Regresión Ridge
# Primera iteración
alphas = np.linspace(0.01, 50.0)
rr = linear_model.RidgeCV(alphas=alphas)
rr.fit(x_train,y_train)
print(rr.alpha_)

# Segunda iteración
alphas = np.linspace(50.0, 100.0)
rr = linear_model.RidgeCV(alphas=alphas)
rr.fit(x_train,y_train)
print(rr.alpha_)

# Tercera iteración
alphas = np.linspace(50.0, 90.0)
rr = linear_model.RidgeCV(alphas=alphas)
rr.fit(x_train,y_train)
print(rr.alpha_)

# Cuarta iteración
alphas = np.linspace(90.0, 100.0)
rr = linear_model.RidgeCV(alphas=alphas)
rr.fit(x_train,y_train)
print(rr.alpha_)

# Quinta iteración
alphas = np.linspace(93.5, 94.0)
rr = linear_model.RidgeCV(alphas=alphas)
rr.fit(x_train,y_train)
print(rr.alpha_)

rr.predict(x_test)
print(rr.score(x_test,y_test))
```

Listing 3.4: Regresión Ridge en sklearn

```
# Creación de la gráfica comparativa
alphas = np.linspace(.1, 120.0,120)
plt.figure()
plot_learning_curve(linear_model.LinearRegression(),"Least Squares", x, y,"r", ylim=(-0.1,
    1.1))
plot_learning_curve(linear_model.RidgeCV(alphas=alphas),"Ridge", x, y, "b",ylim=(-0.1, 1.1))
plt.title("Least squares vs Ridge")
plt.savefig("leastsquares_vs_ridge")
```

Listing 3.5: Creación de la gráfica comparativa entre mínimos cuadrados y Ridge

4

Diseño y desarrollo

En este capítulo se va a explicar detalladamente el software creado, explicando primero las diferentes herramientas y librerías utilizadas y después el proceso de creación del sistema y las decisiones tomadas a lo largo del mismo.

4.1. Herramientas software

Para la realización de este sistema se ha utilizado como lenguaje de programación Python ya que disponemos de la librería `pygrib` que nos permite extraer los datos de los ficheros fácilmente y la librería `pandas` que nos permite guardar los datos en un `Dataframe` facilitándonos la tarea de guardado y tratamiento de los datos extraídos. Otra librería utilizada, aunque de forma más indirecta, es `NumPy`, que es la base de `pandas` y nos ayuda a tratar con `arrays` multidimensionales. Estas librerías han sido utilizadas en capítulos anteriores (en las secciones 2.3 y 3.3.1 y en el anexo A.3), por lo que aquí se realiza una breve explicación.

4.1.1. Librería NumPy

La librería `NumPy` (Numerical Python) es un paquete orientado a la computación científica y el análisis de datos. Está programado en Python y C.

El tipo de datos básico en esta librería son los `arrays`, que aunque a primera vista son como listas de listas, nos aportan una serie de atributos y métodos para trabajar eficazmente con ellos. Posibles ejemplos son:

- Obtener información básica del `array`: esto se puede hacer con los atributos `x.shape`, `x.size` y `x.dtype`. La forma del `array` se puede modificar con el método `x.reshape()` y el tipo con `x.astype()`.
- Valores especiales: los `arrays` pueden contener valores especiales como el infinito y el valor `nan` (*not a number*). Estos valores se encuentran en las variables `numpy.inf` y `numpy.nan`.
- Unir `arrays`: se pueden juntar varios arrays de diferentes formas; verticalmente con `numpy.vstack()`, horizontalmente con `numpy.hstack()` o en función de un eje pasado como parámetro con `numpy.stack()`.

A	1
B	2
C	3
D	4
E	5
F	6
G	7

Cuadro 4.1: Ejemplo `Series`

	Uno	Dos	Tres	Cuatro
A	1	7	5	4
B	2	6	6	3
C	3	5	7	2
D	4	4	1	1
E	5	3	2	7
F	6	2	3	6
G	7	1	4	5

Cuadro 4.2: Ejemplo `Dataframe`

- Guardar los `arrays`: podemos guardar los datos tanto en formato de texto plano con `numpy.savetxt()` como en formato `npy`, el formato propio de la librería, con `numpy.save()`. También disponemos de los formatos `npz`, que nos permite guardar varios `arrays` en un mismo fichero y `npz comprimido`, que utiliza `gzip` para reducir el tamaño de guardado en disco.

- Operaciones matemáticas y estadísticas: algunas operaciones matemáticas se pueden hacer directamente sobre los `arrays`, como la suma (`arrC = arrA + arrB`), la multiplicación término a término (`arrA*3`) o la exponencial (`arrA**2`). Esto también se puede hacer con los métodos de la librería, como `add()`, `exp()`, `sqrt()`.

  También disponemos de métodos estadísticos como `numpy.mean()`, `numpy.var()`, `numpy.min()`, `numpy.std()`.

Además de esto, la librería nos ofrece métodos y submódulos que nos permiten trabajar con histogramas, números aleatorios, álgebra lineal o funciones polinómicas.

4.1.2. Librería pandas

La librería `pandas` nos permite crear dos nuevos tipos de datos: las `Series` y los `Dataframes`.

Las `Series` son `arrays` unidimensionales con un índice asociado. Un ejemplo de `Series` sería el cuadro 4.1. La librería nos aporta una serie de atributos (`size`, `shape`, `values`, etc.) y métodos (`max`, `min`, `add`, `abs`, etc.) que nos permiten trabajar con los datos de las `Series`. La mayoría de estos derivan de la librería `NumPy`.

Para acceder a los datos lo podemos hacer mediante posición numérica con el método `iloc` o mediante las etiquetas del índice con el método `loc`. Los índices también nos permiten seleccionar fácilmente subconjuntos del `Dataframe`, por ejemplo, `s[s.index > 'C']`.

Los `Dataframes` son `arrays` multidimensionales, con un índice para las filas y otro índice para las columnas. Pueden ser vistos como un conjunto de `Series` con un índice común. Un ejemplo de `Dataframe` sería el cuadro 4.2.

Para crearlos, la librería nos da la opción de hacerlo a partir de una lista de `Series`, de un diccionario o a partir de un `array`. La librería también nos aporta una serie de atributos (`size`, `shape`, `values`, etc.) y métodos (`max`, `min`, `add`, `abs`, `join`, `merge`, etc.) que nos permiten trabajar con los datos de los `Dataframes`. La mayoría de estos derivan de la librería `NumPy`.

También disponemos del método `concat` que nos permite concatenar fácilmente cualquier tipo de objeto creado por `pandas`. En función de los argumentos pasados podemos añadir cierta lógica a la concatenación, como por ejemplo tenemos la posibilidad de crear conjuntos de datos a partir de la unión o la intersección de varios objetos.

Formatos de guardado de Dataframes

Tanto para `Dataframes` como para `Series`, la librería nos ofrece métodos para guardar y para recuperar los datos. Para facilitar su uso, todos estos métodos tienen la misma estructura (para guardar `to_'formato'` y para recuperar `read_'formato'`). Los formatos de los que dispone son:

- Formato `txt`: es el formato más simple de todos y legible por humanos. Se trata de texto plano en el que solo se guardan caracteres, no el tipo de datos que contiene.

  Para guardar un `Dataframe` en este formato, `pandas` nos ofrece el método `pandas.to_string()` que nos devuelve el `Dataframe` en una cadena de texto que podemos escribir en un fichero. Alternativamente podemos ir recorriendo el `Dataframe` y guardarlo en un fichero de texto manualmente, estructurándolo como nos sea más conveniente.

- Formato `csv` (comma-separated value): es un formato estandarizado y de formato abierto (no está sujeto a ningún tipo de licencia) que nos permite representar tablas fácilmente. En este formato, las diferentes columnas están separadas por comas (u otro carácter si se indica) y las filas por saltos de línea. Al igual que el formato `txt`, este formato no guarda el tipo de datos que contiene y es legible por humanos.

  Para guardar un `Dataframe` en este formato, lo podemos hacer manualmente recorriendo el `Dataframe` o mediante el uso de los métodos de `pandas`. El uso de estos métodos (`pandas.to_csv()` y `pandas.read_csv()`) es más recomendable ya que no tenemos que preocuparnos por la estructura del fichero y nos aporta algunas opciones de gran utilidad, como la opción de comprimir el fichero creado o parsear los datos del fichero automáticamente.

- Formato `HDF` (Hierarchical Data Format): este formato, creado por el Centro Nacional de Aplicaciones de Supercomputación (NCSA por sus siglas en inglés), también de formato abierto, nos permite guardar el tipo de datos que contiene. Esto es porque lo que hace es serializar los datos, es decir, guardarlos como una cadena de bits para que al volver a leerlos se recupere una copia exacta de los datos.

  Para trabajar con `Dataframes` en este formato contamos con los métodos `pandas.to_hdf()` y `pandas.read_hdf()` que nos permiten guardarlos y recuperarlos respectivamente.

- Formato `pickle`: es el formato propio de Python para serializar objetos. Al igual que pasa con `HDF`, este formato también nos conserva los tipos de datos y la información asociada al objeto serializado.

  Para trabajar con `Dataframes` en este formato podemos hacerlo manualmente mediante la librería propia de Python o automáticamente con el uso de los métodos de la librería `pandas` (`pandas.to_pickle()` y `pandas.read_pickle()`). Si utilizamos una versión de Python superior a la 3.0, también podemos utilizar el módulo `cPickle` que aporta velocidad a la hora de trabajar con ficheros con este formato.

  Este formato no es seguro contra datos mal formados o maliciosos, por lo que hay que tener cuidado a la hora de trabajar con ficheros creados por fuentes no fiables.

- Formato `JSON` (JavaScript Object Notation): es un formato de texto ligero, de formato abierto y estandarizado. Este formato también es legible por humanos. Los datos están organizados como si fuese un diccionario (los campos están guardados en pares clave-valor) por lo que resulta muy fácil parsearlos.

  Para trabajar con `Dataframes` en este formato contamos con los métodos `pandas.to_json()` y `pandas.read_json()` que nos permiten guardarlos y recuperarlos respectivamente.

- Formato `msgpack`: se trata de un formato basado en `JSON` pero sujeto a licencia. Su principal diferencia es que no es legible por humanos, sino que se serializan los datos con el fin de hacerlo más compacto.

  Para trabajar con `Dataframes` en este formato contamos con los métodos `pandas.to_msgpack()` y `pandas.read_msgpack()` que nos permiten guardarlos y recuperarlos respectivamente.

- Además de estos, la librería `pandas` también nos permite guardar los datos en `HTML`, `Microsoft Excel`, `Feather Format`, `Stata`, `SAS`, `SQL` O `Google Big Query`.

Otros formatos en los que se pueden contener los datos son npy/npz, que aunque para guardar los datos de esta manera no existen métodos propios de `pandas`, se han tenido en cuenta por su alta popularidad. Estos formatos son los que utiliza la librería `numpy` para guardar sus `arrays`, y puesto que los datos del `Dataframe` están creados con este tipo de `arrays`, podemos salvarlos en disco en este formato.

Para guardar el `Dataframe` de esta manera, tenemos que guardar los datos, el índice y las cabeceras de las columnas por separado; con el método `dump` de la librería `pickle` para índices y cabeceras y con los métodos `save`, `savez` y `savez_compressed` de la librería `NumPy` para guardar los datos en formato `npy`, `npz` y `npz comprimido` respectivamente.

4.1.3. Librería time

Para facilitar la comparación de los diferentes tipos de formatos en función del tiempo que tardan en procesar los datos, se ha utilizado la librería `time` que nos permite fácilmente tomar tiempos. Esto lo podemos hacer, por ejemplo, con el siguiente código:

```
import time
start_time = time.time()
# código a ejecutar
end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
```

El método `time()` nos devuelve el tiempo en segundos en función del tiempo de referencia de UNIX (1 de enero de 1970 a las 00:00:00 en tiempo UTC). El valor devuelto por esta función es de tipo *float*, por lo que la precisión de la toma de tiempos depende de la máquina en la que se ejecute el código.

4.1.4. Herramienta cron

Puesto que el mantenimiento del `Dataframe` histórico (recepción del fichero GRIB, creación del fichero intermedio y actualización del `Dataframe`) son tareas diarias, se ha utilizado la herramienta de Linux `crontab` para automatizarlas y evitar posibles olvidos. Esta herramienta nos permite programar tareas para un momento dado que se ejecutaran siempre y cuando la máquina esté activa. Para añadir tareas hay que ejecutar el comando `crontab -e` y añadir una línea con el formato

`minutos horas día mes día-de-la-semana comando-a-ejecutar`.

Por ejemplo, para que un comando se ejecute todos los días a las 9:30 sería

```
30 09 * * * comando.
```

Poner un * en uno de los campos significa que ese campo no va a restringir la ejecución del comando. En caso de que el comando ejecutado genere una salida, ya sea de error o salida estándar, si no se ha indicado otra cosa, se envía automáticamente al correo del usuario.

4.1.5. Bases de datos

El manejo de la base de datos se ha realizado con el paquete `psycopg2` que nos permite trabajar con PostgreSQL desde Python de forma sencilla.

En esta librería, los comandos básicos para el manejo de la base de datos son:

- `connect()`: nos permite conectarnos a la base de datos.
- `cursor()`: devuelve un objeto que nos permite realizar las diferentes acciones de modificación de la base de datos.
- `execute()`: ejecuta el comando SQL pasado como parámetro.
- `fetchone()`: si hemos realizado un `SELECT` cada vez que llamamos a este método nos devuelve un diccionario con la información a la siguiente fila de la selección.
- `commit()`: guarda los cambios realizados en la base de datos.
- `rollback()`: si ha habido algún error al guardar los cambios devuelve la base de datos a un estado seguro anterior.
- `close()`: cierra el cursor y la conexión a la base de datos.

4.1.6. Documentación del código

Para mantener el código documentado se ha utilizado la herramienta Sphinx, ya que nos permite automatizar la extracción de los comentarios de dentro del código a documentos separados. Para mantener un control sobre el mismo y poder disponer de la documentación en diferentes formatos (pdf, html, latex) se ha utilizado la herramienta Read the docs. Una explicación detallada del uso de ambas herramientas la podemos encontrar en el apéndice B.

4.1.7. Control de versiones

Para llevar a cabo un control de versiones y mantener actualizada una copia de seguridad de todo el proyecto, se ha utilizado la herramienta git, un software de control de versiones creado por Linus Torvalds en 2005 [5]. Esta plataforma nos facilita tener todos los documentos actualizados y poder acceder a ellos desde distintos equipos y plataformas.

Los principales comandos de git son:

- `git add`: añade los cambios realizados localmente a la lista de ficheros guardar en el repositorio local.
- `git commit`: guarda los ficheros indicados en el repositorio local.
- `git status`: comprueba las diferencias entre la copia local y la copia del servidor.

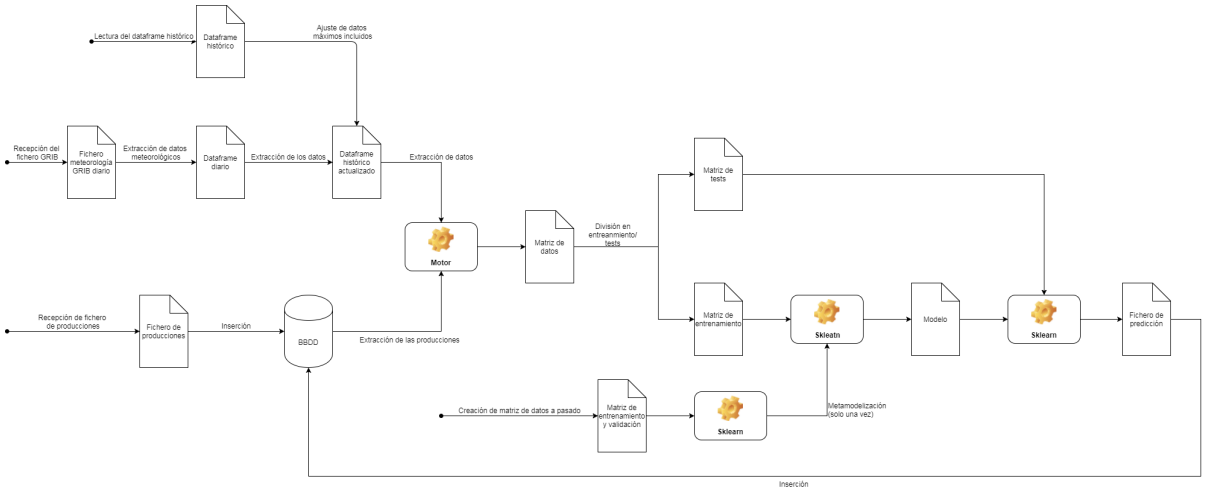

Figura 4.1: Diagrama del proceso de datos

- `git push`: guarda los cambios realizados en el repositorio local en el repositorio remoto.
- `git pull`: descarga los cambios realizados en el repositorio remoto en el repositorio local.

4.2. Diseño

Se recuerda que el software creado tiene como fin la automatización de la generación de un histórico de datos meteorológicos a partir de los cuales, y mediante la adición de datos de predicciones meteorológicas, nos permite crear una matriz de datos (forma de colocar los datos en forma de rejilla de tal manera que las unidades están en las filas y las variables en las columnas). De esta manera facilitamos la posterior utilización de los datos meteorológicos obtenidos para generar modelos de predicción.

Para hacer esto, una vez obtenido un fichero GRIB, extraemos los datos y los guardamos en un fichero intermedio tanto como por tenerlo como respaldo, como por si volvemos a necesitar los datos en un futuro evitar tener que realizar otra vez la extracción de los datos. Después de esto obtenemos los datos que nos interesen y los guardamos en el `Dataframe` correspondiente, lo que nos permite disponer de un histórico de datos actualizado día a día.

Paralelamente, se reciben los datos de producción, los cuales introducimos en una base de datos. Estos registros se recuperan posteriormente al crear la matriz de datos para introducirlos en la misma a la vez que se obtienen los datos meteorológicos del rango de días que se vayan a utilizar en el modelo de predicción.

Con la matriz de datos ya creada, se divide en conjunto de entrenamiento para crear el modelo y conjunto de pruebas para comprobar que el modelo creado funciona correctamente. Podemos ver un gráfico explicativo del proceso en la figura 4.1.

Para confeccionar la matriz de datos y utilizarla para la predicción de datos, se siguen los siguientes pasos:

- Se extraen los datos de los ficheros meteorológicos diarios y se guardan en un diccionario (con el código de la sección 2.3.2). En estos diccionarios discriminaremos los datos primero en función de la fecha de generación, después por la fecha de predicción y por último por nombre y coordenadas de la variable. De esta manera, si en un mismo fichero hay varias predicciones para un mismo día, las mantenemos separadas y evitamos posibles colisiones a la hora de crear el `Dataframe`.

prediction date	generation date	10 metre U wind component(29.5, 341.5)	10 metre U wind component(29.5, 341.625)
2017-09-16 00:00:00	2017-09-16	-9.825193	-9.770506
2017-09-16 03:00:00	2017-09-16	-10.963103	-10.957732
2017-09-16 06:00:00	2017-09-16	-10.809504	-10.775324
2017-09-16 09:00:00	2017-09-16	-10.603947	-10.632511
2017-09-16 12:00:00	2017-09-16	-10.956985	-10.993606
2017-09-16 15:00:00	2017-09-16	-10.570086	-10.551044
2017-09-16 18:00:00	2017-09-16	-10.488659	-10.510143
2017-09-16 21:00:00	2017-09-16	-10.612213	-10.690826
2017-09-17 00:00:00	2017-09-16	-11.093393	-11.107553
2017-09-17 03:00:00	2017-09-16	-11.147841	-11.228408

Cuadro 4.3: Ejemplo parcial de un Dataframe histórico

- Mediante el uso de la librería `pandas` transformamos el diccionario en un `Dataframe` que incluye la fecha de predicción, la fecha de generación de los datos, y los datos en función de la variable que representan y las coordenadas geográficas. Un ejemplo de un `Dataframe` parcial de este tipo sería la tabla 4.3.

- Una vez creados, estos `Dataframes` se guardan en formato `pickle` mediante el método `to_pickle()`. Aunque por defecto los `Dataframes` incluyen los datos de todas las fechas de predicción posibles de los ficheros GRIB, el método que los crea nos permite escoger solo un rango de fechas. De esta manera, si el fichero tiene predicciones a 10 días pero solo nos interesan los 3 primeros podemos evitar el procesamiento de los 7 restantes, ahorrándonos así el tiempo y espacio que ello acarrea.

- Después de esto, añadimos las filas correspondientes al `Dataframe` histórico que vayamos a utilizar para crear la matriz de datos. A la hora de crear estos `Dataframes` históricos, podemos elegir el número de días que hay entre la generación de los datos y la predicción de los mismos.

  Antes de actualizar estos `Dataframes` históricos, comprobamos el rango de fechas incluidos en los mismos, para que , por ejemplo, no contengan más de tres años completos de datos.

- Paralelamente a este proceso, se obtienen los datos de producción, los cuales se guardan en una base de datos. A la hora de crear la matriz de datos, se obtiene la información de producción correspondiente a las fechas del `Dataframe` histórico que se vaya a utilizar en la creación del modelo y se añaden a la misma.

- Con la matriz de datos ya creada, la dividimos en dos subconjuntos, uno para entrenamiento y otro para pruebas. Con el conjunto de entrenamiento y los datos obtenidos de una metamodelización previa, creamos un modelo de predicción con regresión ridge.

- Finalmente, con el modelo creado y el conjunto de pruebas realizamos predicciones y las guardamos en la base de datos.

4.3. Desarrollo

El proceso de desarrollo se ha dividido en tres partes: la obtención y guardado de los datos en un `Dataframe`, la obtención y guardado de las predicciones de energía eólica y la unión de los datos anteriores para generar un modelo de predicción.

Formato	Tamaño descomprimido (MB)	Tiempo de creación de fichero intermedio (segundos)			
		Ejecución 1	Ejecución 2	Ejecución 3	Ejecución 4
hdf5	32,3	61,41	60,92	59,88	60,19
csv	55,0	71,22	70,95	71,74	72,26
pickle	27,4	60,32	67,76	60,50	60,85
msgpack	27,5	60,25	60,26	60,44	60,90

Cuadro 4.4: Comparación de formatos para ficheros intermedios

4.3.1. Manejo de datos meteorológicos

Creación de ficheros intermedios

Para crear este sistema, se empezó con el estudio de la librería `pygrib`, que es la encargada de extraer los datos de los ficheros GRIB (sección 2). Para ello se crearon pequeños programas que leían la información de dichos ficheros y la extraían de tal manera que fuese fácilmente utilizable.

Después de esto, se procedió a buscar una manera de guardar los datos extraídos, de tal manera que fuesen fácilmente reutilizables. En un principio se intentó guardar en formatos legibles, como el `txt` o el `csv` de forma manual, pero el tamaño de los ficheros generados era excesivamente grande y obligaba a tener que seguir un formato no estandarizado, sino creado manualmente, lo que podría aumentar el riesgo de fallos.

Para solucionar esto se procedió al estudio de los diferentes formatos a los que la librería `pandas` permite transformar de forma nativa los `Dataframes` creados. Los formatos estudiados fueron `hdf5, csv, pickle y msgpack`. Para comparar el rendimiento de dichos formatos, se utilizó un mismo fichero (se escogió un fichero GRIB estándar que ocupaba 6,3 MB con los datos meteorológicos de la península Ibérica de 10 días con horizontes de 3 horas para los 4 primeros días y de 6 horas para el resto) para extraer sus datos y guardar el `Dataframe` creado, viendo así cuanto tiempo se tardaba en hacerlo y cuánto ocupaba el fichero resultante. Los resultados de la comparativa los podemos ver en la tabla 4.4.

Tras ver estos resultados, se descartó tanto `hdf5` como `csv` por el tamaño final del archivo. Finalmente, entre los dos restantes se decidió utilizar el formato `pickle` ya que aunque los tiempos eran muy similares, este generaba un fichero ligeramente más pequeño.

Una vez hecho esto, se crearon unos pequeños *scripts* en los que utilizamos el fichero `grib.py`, que pasándole como argumentos `dataframe`, `crearmatriz` o `actualizarmatriz` nos permite automatizar la generación de los ficheros intermedios, la creación de los `Dataframes` históricos y la actualización de los mismos, respectivamente. Estos *scripts* son los siguientes:

- Creación del `Dataframe` histórico:

```
#!/bin/bash
file=/home/pickle/fichero.pickle
output=/home/matriz_datos/
days=0
python /home/src/grib.py crearmatriz -inputfiles $file -output $output -days $days
```

- Creación del fichero intermedio con los datos del GRIB diario:

```
#!/bin/bash
gribfile=/home/GRIB/fichero.grib
output=/home/pickle/
python /home/src/grib.py dataframe -file $gribfile -output $output
```

Formato	Tamaño del Dataframe diario (MB)	Tamaño del Dataframe histórico (MB)
pickle	12.7	1218.56
pickle comprimido	4.50	529
csv comprimido	5.20	606
npy	14.1	1372.16
npz	14.1	1372.16
npz comprimido	6.33	452

Cuadro 4.5: Comparación de tamaños para Dataframes

Formato	Tiempo de creación del dataframe histórico (segundos)	Tiempo de creación del dataframe diario (segundos)	Tiempo de actualización del dataframe histórico (segundos)	Tiempo de actualización diario (segundos)	Tiempo de carga del dataframe histórico en memoria (segundos)	Tiempo total diario (segundos)
pickle	91,73	21,47	5,90	27,37	1,09	28,46
pickle comprimido	1443,96	35,24	1345,70	1380,94	8,23	1389,17
csv comprimido	2249,86	30,82	6,64	37,46	254,20	291,66
npy	906,37	23,22	47,31	70,53	24,08	94,61
npz	996,12	22,55	79,09	101,64	50,33	151,97
npz comprimido	1059,24	23,94	202,18	226,12	59,00	285,12

Cuadro 4.6: Comparación de tiempos para Dataframe

- Actualización del `Dataframe` histórico:

```
#!/bin/bash
inputfiles='/home/pickle/fichero.pickle'
matriz=/home/matriz_datos/matrizdatos
python /home/src/grib.py actualizarmatriz -file $matriz -inputfiles $inputfiles
```

Creación de Dataframes históricos

Una vez tomada la decisión de guardar los ficheros intermedios en formato `pickle`, se crearon `Dataframes` de mayor tamaño con la información meteorológica de varios días para comprobar que al tratar con grandes `Dataframes`, el formato `pickle` seguía siendo la mejor opción. Con este `Dataframe` creado, se observó que aunque los tiempos de tratamiento diario de los ficheros (creación de ficheros intermedios y creación y actualización de `Dataframes` históricos) eran asequibles, el tamaño de los mismos era considerable. Para encontrar una solución a esto, se realizó una comparativa de formatos que permiten comprimir los datos, en concreto, se compararon `csv` y `pickle` comprimiéndolos con `gzip` y los formatos `npy`, `npz` y `npz comprimido`.

Para comparar esos formatos se hizo una batería de pruebas que obtenía los tiempos de creación de ficheros intermedios, de creación un `Dataframe` con once meses de datos, la adición de datos diarios al `Dataframe` histórico creado anteriormente hasta completar un año de datos y la lectura del mismo. Para automatizar esta batería de pruebas se creó un pequeño `script` que iba llamando a los métodos correspondientes. El `script` creado lo podemos encontrar en el anexo C.

Para tomar estos datos, se utilizaron ficheros GRIB de 2,62 MB de tamaño con los datos meteorológicos correspondientes a la península Ibérica de 3 días con horizontes de 3 horas. El `Dataframe` creado contenía los datos meteorológicos de la península desde el 01/01/2016 hasta 31/12/2016.

Los datos obtenidos en función del tamaño los podemos ver en la tabla 4.5 y en función del tiempo en la tabla 4.6. En la tabla de comparación de tiempos, la información dispuesta en las columnas de izquierda a derecha es:

- Tiempo que se tarda en crear el `Dataframe` correspondiente a 11 meses desde cero. Aquí, mediante el uso del código de la sección 2.3.2 cargamos los datos de todos los GRIBS en un diccionario. En este apartado utilizamos los datos de 11 meses para que tras añadir los datos diarios de un mes más el `Dataframe` histórico final sea de un año justo. Después lo convertimos en un `Dataframe` y lo guardamos en disco.

- Tiempo que se tarda en obtener los datos de un fichero GRIB de un día. Aquí volvemos a utilizar el código de la sección 2.3.2 para extraer los datos de un GRIB diario. Para calcular este tiempo se ha tomado la media de extracción de los datos de 31 días por separado.

- Tiempo medio que se tarda en incluir al `Dataframe` histórico los datos de un día teniendo en cuenta la lectura del mismo, la adición de los datos y el guardado en disco. En este apartado destaca el formato csv comprimido ya que aunque tarda mucho en leerse tarda muy poco en actualizarse. Esto se debe a que en este formato no hace falta cargar todo el `Dataframe` en memoria para añadir nuevos datos, pues podemos incluirlos abriendo el csv comprimido en modo actualización
  (`df.to_csv(matriz_datos, compression='gzip', mode='a', header=False)`)
  añadiendo así los registros sin necesidad de descomprimir y volver a comprimir. También destaca `pickle comprimido` ya que aunque tarda muy poco en cargar el `Dataframe` histórico en memoria tarda mucho en actualizarlo. Esto se debe a que tiene un alto grado de compresión, por lo que al guardarlo de nuevo en disco tiene que repetir todo el proceso de nuevo resultando esto en un alto coste temporal.

- Suma del tiempo de obtención de los datos de un día y el tiempo que se tarda en incluirlo en el `Dataframe` histórico.

- Tiempo que se tarda en cargar el `Dataframe` histórico en memoria para poder trabajar con él. En este apartado destaca el tiempo de lectura en formato csv comprimido que tarda mucho más que el resto. Esto se debe a que para cargarlo primero lo descomprime y al estar guardado en texto plano tarda más que en los casos que los datos están guardados como cadena de bits.

- Suma del tiempo de obtención de los datos de un día, el tiempo que se tarda en incluirlo en el `Dataframe` histórico y el tiempo que se tarda en cargarlo en memoria para poder trabajar con él.

A la hora de elegir uno de los formatos, se ha tenido en cuenta en primer lugar el tiempo utilizado a diario para actualizar el `Dataframe` y en segundo lugar el tamaño del mismo. Atendiendo a esto, podemos observar dos grupos:

1. Los óptimos en función de tiempo serían los formatos `pickle`, `npy` y `npz`. De estos, podemos escoger como mejor opción `pickle` ya que tarda entre 5 y 10 veces menos que las otras opciones.

2. Los óptimos en función del tamaño serían los formatos `pickle comprimido`, `csv comprimido` y `npz comprimido`. De estos, `pickle comprimido` lo podemos descartar fácilmente ya que tarda más de 10 minutos en realizar las tareas diarias. De los otros dos, aunque son muy similares, se ha escogido como mejor opción `csv comprimido` ya que la actualización del `Dataframe` histórico es mucho menor y esta es una tarea que se realiza a diario.

Finalmente, entre las dos posibles opciones (`pickle` y `csv comprimido`) se eligió la primera ya que aunque el tamaño del `Dataframe` histórico es el doble, el tiempo diario necesario para que este esté actualizado es 10 veces menor con `pickle`.

4.3.2. Manejo de producciones energéticas

Para mantener un registro de las producciones y predicciones que se iban a usar para la creación del modelo, se ha creado una pequeña base de datos que contiene los datos obtenidos de un parque eólico para las zonas incluidas en la matriz de datos. Estas producciones se registran de forma horaria, por lo que en la base de datos se guarda la fecha en la que se han tomado los datos (tanto en formato UTC como en formato de hora legal española). También se guardan las potencias estimada e instalada y el porcentaje de equivalencia de la primera en función de la segunda.

A la hora de crear esta base de datos, el principal problema que hubo que enfrentar fue la gestión de los cambios de hora de verano e invierno. Esto resultaba un problema porque hay un día al año en el que únicamente se reciben datos de 23 horas, y otro en el que se recibían datos de 25 horas, lo que resulta en un conflicto temporal en algunos casos. Para solucionar esto, se utilizó la librería de Python `pytz` que contiene una lista con los cambios de hora. Comprobando en esta lista si la hora que se va a introducir corresponde a un cambio de hora nos permite ajustar manualmente dichos tiempos para evitar los saltos temporales.

4.3.3. Creación del modelo predictivo

Para poder hacer predicciones con los datos se empezó estudiando la librería `sklearn` y los métodos de mínimos cuadrados ordinarios y regresión Ridge para la creación de modelos, como podemos ver en la sección 3.

Una vez hecho esto se procedió a crear el código que nos permitía montar la matriz de datos, cogiendo los datos que nos interesasen del `Dataframe` histórico y las producciones correspondientes de la base de datos. Con estos datos unidos, buscamos el mejor valor alpha posible para un modelo Ridge mediante validación cruzada, como podemos ver el la sección 4.4. Finalmente, se creó un modelo Ridge con el alpha calculado anteriormente que nos permitía realizar predicciones con los datos meteorológicos que se recibían diariamente, las cuales una vez realizadas se guardaban en la base de datos.

4.4. Experimentos realizados

Para comprobar la eficacia del sistema creado, se ha creado un script que realiza las siguientes acciones:

- Crear dos matrices de datos, una con la información meteorológica de 2015 y 2016 para entrenar el modelo y otra con la de 2017 para realizar pruebas. Para ello extraemos de la base de datos las producciones para estos periodos y las añadimos a los `Dataframes` históricos, como podemos ver en el listing 4.1.
- Hyperparametrización de α para un modelo Ridge. Para ello, lo primero que debemos hacer es escalar los datos, para que el modelo sea más eficiente (sea más rápido y obtenga mejores resultados). Esto lo podemos hacer transformando los datos de tal manera que la media sea 0 y la varianza 1.

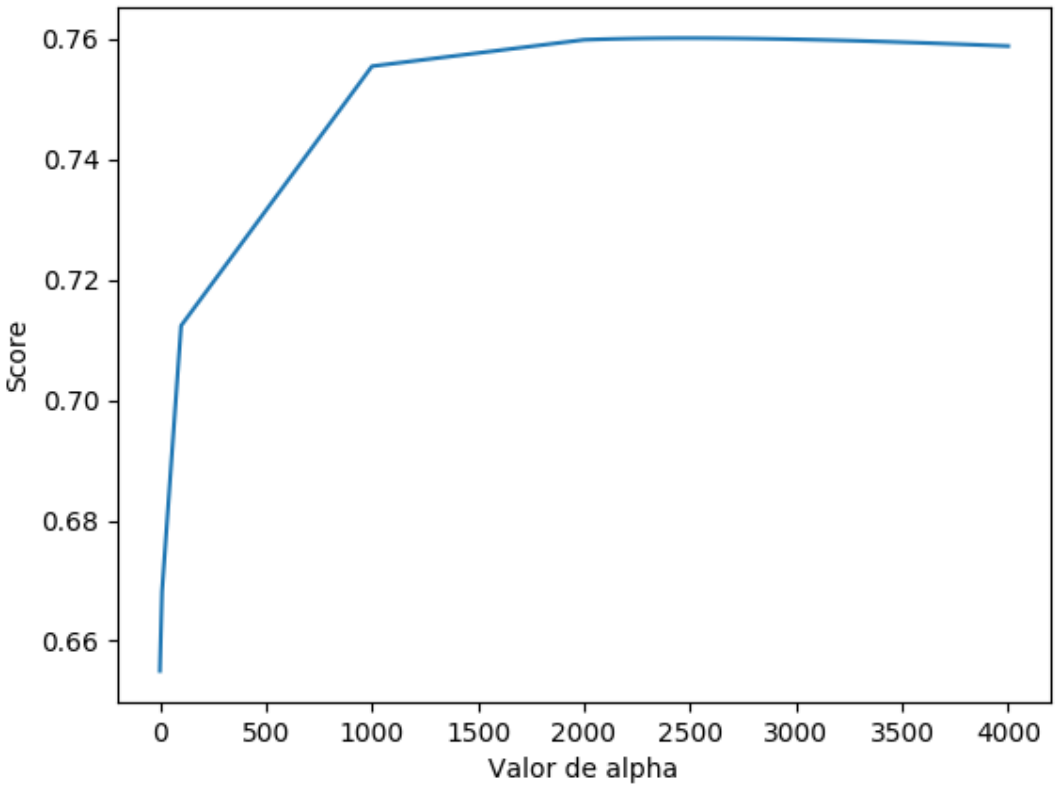

Figura 4.2: Comparativa de alphas

Una vez que tenemos los datos preparados, podemos pasar al cálculo del valor óptimo de alpha. Esto lo hacemos mediante un sistema de validación cruzada con datos de 2 años (2015 y 2016) divididos en 24 partes iguales, es decir, buscamos el mejor valor de α realizando pruebas de entrenamiento-predicción con 23 meses y un mes respectivamente para los 24 meses de datos. Este proceso lo podemos ver en el listing 4.2. Al realizar esta tarea, obtenemos para cada alpha probado, en este caso con valores entre 10^{-6} y 4000, una puntuación o score (ver sección 3.3) entre 0 y 1, como podemos ver en la figura 4.2.

- Creación de un modelo Ridge con el mejor valor de α hallado anteriormente, es decir, el que tenga la puntuación mas cercana a 1. Este modelo lo entrenamos nuevamente con los datos de 2015 y 2016 y probamos su eficacia con los datos de 2017, como podemos ver en el listing 4.3.

  Para comprobar que este modelo es eficiente y las predicciones realizadas son fiables, nos hemos fijado en las medidas del coeficiente de determinación y el error absoluto medio, que en este caso nos dan unos valores de 0,82 y 4,16 respectivamente.

```
import psycopg2
import os
import matplotlib
import pickle
import numpy as np
import pandas as pd
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model, model_selection, preprocessing, metrics

def add_producciones(df):
    """
    Añade los datos de produccion al Dataframe pasado como parámetro.

    Crea un Dataframe que contiene los datos meterológicos y las
    producciones asociadas a las fechas de los mismos.

    :param df: Dataframe al que añadir las producciones
    :return: Dataframe con los datos meteorológicos y las producciones asociadas.
    None si ha habido algún problema con la base de datos.
    """

        con, cur = connect_db()

        comando = "SELECT fecha_legal,estimada,instalada,porcentaje" \
        " FROM producciones"

        cur.execute(comando)
        data = cur.fetchall()
        d = pd.DataFrame(data)

        df_prod = pd.DataFrame(data=d[3].values,index=d[0].values,columns=['Potencia'])
        df_merged = pd.merge(df_prod, df, left_index=True, right_index=True, how='inner')

        close_db(cur, con)
        return df_merged

def montar_matriz_datos(df):
    """
    Añade las producciones correspondientes a las fechas incluidas en
    el dataframe pasado como argumento.

    Devuelve un dataframe con las producciones relativas a las fechas
    incluidas en el dataframe pasado como argumento en la última columna.

    :param df: dataframe con los datos meteorológicos
    :return: dataframe con los datos de producción añadidos
    """

    df = add_producciones(df)
    l = df.columns[1:].tolist()
    l.append(df.columns[0])
    df = df[l]

    return df

df = pd.read_pickle('historico_2015-2017')
df_17 = df[df.index.year==2017]
df = df[df.index.year<2017]
df = montar_matriz_datos(df)
df_17 = montar_matriz_datos(df_17)
```

Listing 4.1: Creación de la matriz de datos

```
# Escalado de los datos
std_scaler = preprocessing.StandardScaler()
x_train = std_scaler.fit_transform(df[df.columns[:-2]])
y_train = df[df.columns[-1]]

x_test = std_scaler.transform(df_17[df_17.columns[:-2]])
y_test = df_17[df_17.columns[-1]]

# Cálculo del alpha óptimo
l_alpha = [10.**k for k in range(-6, 4)] + list(np.linspace(2000,4000))
lr_m = linear_model.Ridge()
param_grid = {'alpha': l_alpha}
modelo = model_selection.GridSearchCV(lr_m, param_grid=param_grid, cv=24,scoring='r2',verbose
    =0)
modelo.fit(x_train, y_train)
```

Listing 4.2: Obtención del valor óptimo de alpha

```
ridge = linear_model.Ridge(alpha=modelo.best_estimator_.alpha)
ridge.fit(x_test,y_test)
pred = ridge.predict(x_test)
print("Score: ",ridge.score(x_test,y_test))
print("Mean absolute error: ",metrics.mean_absolute_error(y_test,pred))

pickle.dump(ridge, open("ridge_2015-2016", 'wb'))
```

Listing 4.3: Creación del modelo ridge óptimo

5
Conclusiones

A lo largo de este documento se han descrito los pasos seguidos para la aplicación de aprendizaje automático en Python, en este caso para que dadas unas predicciones meteorológicas, poder predecir a partir de las mismas la cantidad de energía que se va a producir. Para realizar esto se ha creado un sistema en el que se simulan los pasos que se realizan desde la recepción de los datos meteorológicos hasta la obtención de las predicciones. En todo momento se ha pretendido que este sistema sea lo más óptimo posible, estudiando las diferentes opciones disponibles. Para que esto sea así nos hemos centrado en los siguientes puntos:

- Recepción de los datos meteorológicos: se han estudiado dos tipos de ficheros utilizados para el guardado de datos meteorológicos, GRIB y netCDF, y aunque ambos pueden ser utilizados de igual manera, se ha elegido el primero de estos por facilidad de obtención y disponibilidad de los mismos.
- Tratamiento de los datos meteorológicos: esto se ha realizado en dos fases, la extracción y guardado de los datos diarios, y la creación de un histórico de datos meteorológicos. En ambos casos se ha realizado un estudio exhaustivo de las posibles maneras de guardar estos datos una vez procesados, atendiendo principalmente al tiempo de guardado/recuperación de los datos y al tamaño en disco de los mismos. Teniendo en cuenta que se le dio más importancia al tiempo de ejecución que al espacio ocupado por los datos, se concluyó que la mejor forma de guardar este tipo de datos era usando el formato `pickle` mediante los métodos propios de la librería de Python `pandas`.
- Creación de un modelo predictivo: una vez confeccionada la matriz de datos, se han estudiado los métodos de aplicación de aprendizaje automático en Python mediante la librería `sklearn`. Específicamente se han estudiado los métodos de regresión lineal y se ha aplicado un modelo Ridge, en el que hemos conseguido un coeficiente de determinación de 0,82 y un error absoluto medio de 4,16.

Glosario de acrónimos

- **ASCII**: American Standard Code for Information Interchange
- **CSV**: Comma separated Value
- **ECMWF**: European Centre for Medium-Range Weather Forecasts
- **GRIB**: GRIdded Binary
- **HDF**: Hierarchical Data Format
- **IIC**: Instituto de Ingeniería del Conocimiento
- **JPEG**: Joint Photographic Experts Group
- **JSON**: JavaScript Object Notation
- **NCEP**: National Centers for Environmental Prediction
- **NCSA**: National Center for Supercomputing Applications
- **PNG**: Portable Network Graphics
- **UCAR**: University Corporation for Atmospheric Research
- **UTC**: Tiempo Universal Coordinado
- **WMO**: World Meteorological Organization

Bibliografía

[1] WMO (World meteorological Organization). Documentación oficial de grib. https://www.wmo.int/pages/prog/www/WDM/Guides/Guide-binary-2.html.

[2] ECWMF (European Centre for Medium-Range Weather Forecasts). Página oficial de la grib_api. https://software.ecmwf.int/wiki/display/GRIB/Home.

[3] ECWMF (European Centre for Medium-Range Weather Forecasts). Página oficial de eccodes. https://software.ecmwf.int/wiki/display/ECC/What+is+ecCodes.

[4] Unidata. Página oficial de netcdf. http://www.unidata.ucar.edu/software/netcdf/docs/index.html, Noviembre 2016.

[5] Software Freedom Conservancy. Página oficial de git. https://git-scm.com/.

[6] jswhit. Repositorio oficial de pygrib. https://github.com/jswhit/pygrib/tree/master/docs, Diciembre 2014.

[7] jswhit. Ejemplo de uso de pygrib de la documentación oficial. http://nbviewer.jupyter.org/gist/jswhit/8635665, Enero 2016.

[8] Unidata. Repositorio oficial de netcdf4. https://github.com/Unidata/netcdf4-python, Septiembre 2017.

[9] Unidata. Documentación oficial de netcdf4. http://unidata.github.io/netcdf4-python/, Septiembre 2017.

[10] Unidata. Ejemplos de uso de netcdf. https://www.unidata.ucar.edu/software/netcdf/examples/programs/, Marzo 2014.

[11] Georg Brandl and the Sphinx team. Página oficial de sphinx. http://www.sphinx-doc.org/es/stable/index.html.

[12] Georg Brandl and the Sphinx team. Tutorial de la página oficial de sphinx. http://www.sphinx-doc.org/es/stable/tutorial.html.

[13] Andrew Carter. Guía de documentación con sphinx. https://pythonhosted.org/an_example_pypi_project/sphinx.html, 2009.

[14] Sam Nicholls. An idiot's guide to python documentation with sphinx and readthedocs. https://samnicholls.net/2016/06/15/how-to-sphinx-readthedocs/, Junio 2016.

[15] Inc Read the Docs and contributors. Guía oficial de read the docs. http://docs.readthedocs.io/en/latest/getting_started.html, 2017.

[16] Wes McKinney and PyData Development Team. Documentación oficial de pandas. https://pandas.pydata.org/pandas-docs/stable/index.html, Octubre 2017.

[17] José Dorronsoro. Basic introduction to python's basics. Technical report, Escuela Politécnica Superior,Universidad Autónoma de Madrid, IIC, Julio 2017.

[18] Jerome Friedman Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning.* Springer, 2008.

[19] scikit-learn developers. Documentación oficial de scikit-learn. http://scikit-learn.org/stable/documentation.html, Octubre 2017.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

A

Librería netCDF4

A.1. Ficheros netCDF

Es un conjunto de librerías y formato de datos caracterizado por ser autodescriptivo, multiplataforma y por permitir guardar los datos en forma de array de una forma clara y sencilla. NetCDF fue creado por Unidata, uno de los programas creados por la University Corporation for Atmospheric Research (UCAR), para tratar con datos de carácter científico de una forma eficaz. Aunque está creado sobre librerías de C, hay interfaces que permiten su uso en multitud de lenguajes, como java, Python, Ruby, R, Matlab, etc [4]. Los ficheros netCDF están compuestos por:

- Grupos: estos se asemejan a un sistema de ficheros ya que son estructuras que pueden contener dimensiones, variables, atributos e incluso otros grupos. Al crear un fichero se crea un grupo raíz del que dependerán el resto de elementos que se vayan creando.
- Dimensiones: representan el tamaño de las variables que se vayan a crear. Están definidas por un nombre y un tamaño, que indica la longitud máxima que tendrá el array representado por la variable definida por dicha dimensión. Existe un tipo especial de dimensión llamada dimensión ilimitada, que no tiene tamaño fijo y que va aumentando su tamaño según se van introduciendo datos a la variable a la que define.
- Variables: se utilizan para guardar los datos del fichero. Son arrays con su tamaño indicado por las dimensiones que la definen. A las variables que no tienen ninguna dimensión se las llama escalares y están representadas como arrays de tamaño 0. Las variables pueden ser números en coma flotante, enteros con signo, enteros sin signo y cadenas de caracteres.
- Atributos: aportan información sobre los diferentes grupos o variables. Pueden ser números, cadenas de caracteres o listas.
- Tipos de datos compuestos: son el equivalente a las estructuras en C. Nos permiten tener variables con varios campos, que pueden ser variables u otros tipos compuestos.
- Tipos de datos de longitud variable: se trata de los denominados "ragged" arrays, es decir, arrays en los que las filas no tienen por qué tener todas las mismas longitudes.

```
netcdf test {
dimensions:
        x = 5 ;
        y = 5 ;
        unlim = UNLIMITED ; // (6 currently)
variables:
        int data(x, y) ;
                data :descripcion = "ejemplo de atributo en una variable" ;
        int unlim_data(unlim, x) ;

// global attributes:
                :atributo = "ejemplo de atributo en un grupo" ;
                :atributo2 = "Otro ejemplo de atributo" ;
data:

 data =
  1, 2, 3, 4, 5,
  6, 7, 8, 9, 10,
  11, 12, 13, 14, 15,
  16, 17, 18, 19, 20,
  21, 22, 23, 24, 25 ;

 unlim_data =
  1, 2, 3, 4, 5,
  6, 7, 8, 9, 10,
  11, 12, 13, 14, 15,
  16, 17, 18, 19, 20,
  21, 22, 23, 24, 25,
  26, 27, 28, 29, 30 ;

group: Grupo1 {
  variables:
        int escalar ;
  data:

   escalar = 5 ;

  group: Grupo1.1 {
    } // group Grupo1.1
  } // group Grupo1

group: Grupo2 {

  group: Grupo2.2 {
    } // group Grupo2.2
  } // group Grupo2
}
```

Listing A.1: Fichero netCDF descomprimido

- Tipos de datos enumerados: permiten crear enumeraciones personalizadas.

Si descomprimimos un fichero netCDF (mediante el comando `ncdump fichero.nc` que viene incluido al instalar la librería netCDF4) a formato de texto, podemos observar su composición y estructura. En el listing A.1 podemos observar un fichero netCDF descomprimido que contiene ejemplos de los principales elementos disponibles.

A.2. Librería netCDF4

NetCDF4 es la librería Python que permite el manejo de ficheros netCDF. Se trata de una interfaz de la librería netCDF de C [8].

A.2.1. Descarga e instalación

Esta librería está disponible tanto para entornos Windows como para entornos UNIX. Para obtenerla, la manera más sencilla es mediante la plataforma anaconda con el comando

```
conda install -c anaconda netcdf4.
```

Esta opción está disponible para ambos entornos. Alternativamente, la librería podemos conseguirla de las siguientes maneras:

- Desde Windows:
  - Desde la página oficial: en la página oficial de la UCAR se puede encontrar el instalador de la librería. Una vez descargado el ejecutable se ejecuta y se siguen las instrucciones.
- Desde UNIX:
  - Desde github: para instalar la libraría desde aquí hay que realizar las siguientes acciones:
    1. Descargar o clonar los ficheros desde el repositorio de GitHub.
    2. Ejecutar el comando `python setup.py build`.
    3. Ejecutar el comando `python setup.py install`, con permisos de superusuario si fuese necesario.
    4. Ejecutar el comando `python run_all.py` desde la carpeta `test` para comprobar que se ha instalado correctamente.
       Además del código, en el repositorio de github podemos encontrar ejemplos de uso y documentación sobre la librería.
  - Mediante el comando pip: desde la consola de comandos es suficiente con usar el comando

    ```
    pip install netCDF4.
    ```

    El comando pip se instala automáticamente al instalar Python.

A.2.2. Clases, variables y métodos

En Python, para tratar los ficheros netCDF, disponemos de varias subclases, como por ejemplo la clase `Dataset`, que nos permite crear o abrir los ficheros netCDF y tratar con sus elementos, y una clase por cada uno los objetos que podemos crear (grupos, dimensiones, variables, etc).

Clase `Dataset`

Esta clase es la que nos permite abrir los ficheros netCDF con los que vamos a trabajar. Una vez que lo abre, crea uno grupo raíz a partir del cual podemos manejar los diferentes elementos (grupos, variables, dimensiones, tipos de datos compuestos, tipos de datos de longitud variable y enumeraciones), ya sea para obtenerlos o para crearlos. Estos elementos los guarda en diccionarios ordenados. Además de esto podemos encontrar una serie de atributos que nos aportan información sobre el fichero y los datos:

- `data_model`: indica la versión del modelo de datos del fichero.

- `file_format`: es un alias de `data_model`, por lo que devuelve lo mismo. Se mantiene por motivos de compatibilidad.
- `path`: devuelve una cadena de caracteres con la ubicación del grupo con respecto al grupo raíz.
- `disk_format`: indica el formato del fichero.
- `keepweakref`: indica si las referencias a otros objetos son débiles o no, es decir, si el recolector de basura puede eliminar dichas referencias o no.

En esta clase disponemos de los siguientes métodos:

- `close()`: cierra el fichero netCDF.
- `createCompoundType()`, `createDimension()`, `createEnumType()`, `createGroup()`, `createVariable()`, `createVLType()`: nos permiten crear los diferentes elementos que componen el grupo. Una vez creado uno de estos elementos en un fichero netCDF, no hay forma directa de eliminarlos; para hacerlo debemos copiar todos los datos del fichero excepto el elemento a eliminar.
- `get_variables_by_attributes()`: devuelve una lista con las variables cuyos atributos coinciden con la condición pasada como argumento.
- `ncattrs()`: devuelve una lista con el nombre de los atributos del fichero.

Clase `group`

Esta clase hereda de `Dataset`, por lo que tiene la misma funcionalidad y mismos atributos que su clase padre. Con ella podemos crear grupos, que al igual que su clase padre pueden contener otros grupos, variables, dimensiones, etc.

En esta clase podemos encontrar de utilidad las variables `parent`, que nos devuelve el grupo que lo contiene, y `name`, que nos devuelve el nombre del grupo. Además, en esta clase se redefine el método `close()` para que no se pueda cerrar un grupo, sino que salte una excepción. El resto de variables y métodos funcionan igual que en la clase `Dataset`.

Clase `dimension`

Esta clase nos permite obtener la información asociada a las dimensiones que hayamos creado. Están disponibles las siguientes variables:

- `name`: devuelve el nombre de la dimensión.
- `size`: devuelve el tamaño de la dimensión.

También tenemos los siguientes métodos:

- `group()`: nos indica a qué grupo pertenece.
- `isunlimited()`: nos indica si la dimensión es ilimitada o no.

Clase `variable`

Nos permite trabajar con los datos contenidos en el fichero netCDF. Son equivalentes a los arrays de `numpy`, por lo que podemos acceder a sus datos tanto por índices (`data[5]`) como usando *slicing* (`data[2:7]`). En esta clase disponemos de las siguientes variables:

- `dimensions`: contiene una tupla con las dimensiones que la representan.
- `dtype`: nos indica el tipo de datos que contiene la variable.
- `ndim`: nos indica el número de dimensiones por las que está definida.
- `shape`: contiene una tupla con los tamaños máximos de las diferentes dimensiones.
- `name`: nombre de la variable.
- `size`: número de datos que contiene la variable.

También disponemos de los siguientes métodos:

- `group()`: nos indica a que grupo pertenece.
- `ncattrs()`: devuelve una lista con el nombre de los atributos de la variable.
- `assignValue(val)`: modifica el valor de una variable si ésta es escalar.
- `getValue()`: devuelve el valor de la variable si ésta es escalar.

Clases `compoundType`, `vlType` y `enumType`

Estas clases son las que se encargan de crear los objetos que describen el tipo compuesto, tipo de longitud variable o enumeración que vamos a crear. Estas clases se encargan de crear los descriptores de los objetos, los cuales necesitaremos a la hora de crear las variables, aunque para hacerlo debemos utilizar los métodos `createCompoundType()`, `createVLType()` o `createEnumType()` de la clase `Dataset`.

Atributos asociados a los diferentes elementos de la librería

Los atributos pueden estar asociados al `Dataset`, a grupos o a variables. Para manejar los atributos en estas clases, todas tienen los siguientes métodos:

- `setncattr(name, value)`: nos permite crear un atributo con su descripción. Si el nombre del atributo a crear no es igual a un nombre reservado, también se pueden crear de la siguiente manera:

  ```
  grupo.atributo1 = 'Ejemplo de atributo'.
  ```

- `delncattr(name, attr)`: nos permite borrar un atributo.
- `getncattr(name)`: nos permite obtener la descripción de un atributo a partir de su nombre.
- `renameAttribute(oldname,newname)`: nos permite cambiar el nombre de un atributo ya creado.

A.3. Ejemplos de uso

A.3.1. Ejemplo de creación de ficheros netCDF

En el listing A.2 podemos ver un ejemplo de escritura sobre un fichero netCDF. En él, vemos cómo primero creamos varios grupos de diferentes formas, desde el grupo raíz y desde un grupo ya creado. Después de esto creamos varias dimensiones y variables, a las que finalmente se les añaden atributos.

A.3.2. Ejemplo de obtención de datos de ficheros netCDF

En el listing A.3 encontramos un ejemplo de lectura de un fichero netCDF en el cuál podemos ver cómo primero abrimos el fichero y luego procedemos a la obtención de los diferentes elementos. En primer lugar obtenemos los grupos que hay y su información y atributos. Después, extraemos la información de las dimensiones del grupo raíz y finalmente obtenemos las variables y los datos guardados en las mismas.

A.3.3. Ejemplo de conversión de netCDF a Dataframe

En el listing A.4, vamos obteniendo los datos del fichero netCDF para ir metiéndolos en un diccionario, el cual utilizaremos para volcar fácilmente todos los valores en un `dataframe`. Este `dataframe` tiene como índice la fecha en la que se han tomado las medidas y como columnas el nombre de la variable y las coordenadas de los datos. La creación y manejo de los `dataframes` lo hacemos con la librería `pandas` (sección 4.1.2).

```
import netCDF4 as nc
import numpy as np

# Creamos un fichero con nombre test.nc
group = nc.Dataset("test.nc", "w")

# Creamos dos grupos desde el grupo raíz
g1 = group.createGroup("Grupo1")
g2 = group.createGroup("Grupo2")

# Creamos un grupo en el grupo1 desde el grupo raiz
group.createGroup("Grupo1/Grupo1.1")

# Creamos un grupo en el grupo2 desde él mismo
g2.createGroup("Grupo2.2")

#Creamos dos dimensiones de tamaño 5 y otra ilimitada
x_dim = group.createDimension("x",5)
y_dim = group.createDimension("y",5)
unlim_dim = group.createDimension("unlim",None)

# Variable que representa una matriz de 5x5
data = group.createVariable("data","i4",("x","y"))

#Variable que representa una matriz de 5 columnas y tantas filas como queramos
unlim_data = group.createVariable("unlim_data","i4",("unlim","x"))
escalar = g1.createVariable("escalar","i4")

# Creamos una lista de números del 1 al 26 que luego introduciremos en las variables
aux=np.arange(1,26,dtype="int32")

# Introducimos los datos en la variable data
data[:] = np.reshape(aux,(len(x_dim),len(y_dim)))

# Introducimos los datos en la variable unlim_data
unlim_data[:] = np.reshape(aux,(len(x_dim),len(y_dim)))
# Añadimos una lista de números a la variable unlim_data
unlim_data[5] = np.arange(26,31,dtype="int32")

# Asignamos un valor a la variable escalar
escalar.assignValue(5)

# Añadimos atributos tanto a un grupo como a una variable
group.atributo = "ejemplo de atributo en un grupo"
group.atributo2 = "Otro ejemplo de atributo"
data.descripcion = "ejemplo de atributo en una variable"

group.close()
```

Listing A.2: Ejemplo de creación de ficheros netCDF

```
import netCDF4 as nc


group = nc.Dataset('test.nc', 'r')

# Imprimimos la información básica del grupo raíz
print(group)
print("version del modelo de datos del fichero: ", group.data_model)

print("\nGrupos del grupo raiz:")
for g in group.groups:
    gr = group.groups[g]
    # Imprimimos la información básica del grupo
    print(gr)
    print("Nombre del grupo: ", gr.name)
    print("Grupo padre: ", gr.parent)

print("Atributos del grupo raiz:")
for attr in group.ncattrs():
    print('{}: {}'.format(attr,group.getncattr(attr)))

print("\nDimensiones del grupo raiz:")
for d in group.dimensions:
    dim = group.dimensions[d]
    print(dim)
    print("\tNombre de la dimension: ",dim.name)
    print("\tTamaño de la dimension: ", dim.size)
    print("\tEs una dimension ilimitada: {}\n".format(dim.isunlimited()))

print("Variables del grupo raiz:")
for v in group.variables:
    var = group.variables[v]
    print('Nombre de la variable: {}\n{}'.format(v,var))
    print("Dimensiones de la variable: ", var.dimensions)
    print("Tipo de datos de la variable: ", var.dtype)
    print("Número de dimensiones de la variable: ", var.ndim)
    print("Tamaño de la variable: ", var.shape)
    print("Número de datos de la variable: ", var.size)
    print('Datos de la variable:\n{}\n'.format(group.variables[v][:]))
```

Listing A.3: Ejemplo de obtención de datos de ficheros netCDF

```
import datetime
import netCDF4 as nc
import pandas as pd

offset_date = datetime.datetime(1900, 1, 1, 0, 0)

def netcdf_to_df(netcdf_file=None, forecasted_day=None,output='./'):
    """
    Introduce los datos de un fichero netCDF en un dataframe

    Los datos de la medición se introducen en un dataframe con la fecha
    de la predicción como índice.

    :param netcdf_file: nombre del fichero del que tomar los datos
    :param forecasted_day: número de días a incluir en el diccionario
                           None -> todos los días (valor por defecto)
                           0 -> Día de la toma de datos
                           1 -> Día de la toma de datos más un día
    :return: dataframe con los datos
    """

    group = nc.Dataset(netcdf_file, "r")

    forecast_date = offset_date + datetime.timedelta(hours=int(group.variables['time'][0]))
    lat_shape = group.variables['latitude'].size
    lon_shape = group.variables['longitude'].size

    dicc = {}
    latitudes = group.variables['latitude'][:]
    longitudes = group.variables['longitude'][:]

    # comproboación del rango de datos a incluir en el diccionario
    if forecasted_day != None:
        time_max = forecast_date + datetime.timedelta(days=forecasted_day)

    # Introducimos los datos de los mensajes en un diccionario
    for t, time in enumerate(group.variables['time'][:]):
        for v in group.variables:
            if v == 'time' or v == 'longitude' or v == 'latitude':
                continue
            variable = group.variables[v]
            date = offset_date + datetime.timedelta(hours=int(time))
            if forecasted_day != None and date.day > time_max.day:
                continue
            if not date in dicc:
                dicc[date] = {}
            for i in range(lat_shape):
                for j in range(lon_shape):
                    coords = latitudes[i], longitudes[j]
                    key = variable.long_name + str(coords)
                    dicc[date][key] = variable[t][i][j]

    # Creamos un dataframe con el diccionario y lo guardamos
    df = pd.DataFrame.from_dict(dicc, orient='index')
    date = offset_date + datetime.timedelta(hours=int(group.variables['time'][0]))
    df.insert(0, 'generation day', str(date))
    fecha = date.strftime('%Y%m%d')
    df.to_pickle(output+'netcdf_dataframe_' + fecha + '_pickle')

    return df
```

Listing A.4: Ejemplo de conversión de netCDF a Dataframe

B

Sphinx y Read The Docs

B.1. Sphinx

Sphinx es una herramienta creada por Georg Brandl en 2008 que sirve de apoyo a la documentación de código para diversos lenguajes de programación, aunque el proyecto original se creó pensando en Python [11]. Sphinx nos permite crear documentación en diferentes formatos (HTML, PDF, ePub, latex) a partir de ficheros `reStructuredText` (con extensión `.rst`) y desde la propia documentación del código.

B.1.1. Instalación y creación de un proyecto

Para poder usar Sphinx, es necesario tener instalado Python, por lo que la instalación la podemos realizar con la herramienta `pip` que viene por defecto al instalar Python. El comando a ejecutar sería `pip instal sphinx`.

Para crear un proyecto Sphinx, hay que ejecutar el comando `sphinx-quickstart` desde donde queramos crearlo. Una vez hecho esto nos hará las siguientes preguntas:

- `Root path for the documentation [.]:`
- `Separate source and build directories (y/n)[n]:`
- `Name prefix for templates and static dir [_]:`
- `Project name:`
- `Author name(s):`
- `Project version []:`
- `Project release []:`
- `Project language [en]:`
- `Source file suffix [.rst]:`

- Name of your master document (without suffix)[index]:
- Do you want to use the epub builder (y/n)[n]:
- autodoc: automatically insert docstrings from modules (y/n)[n]:
- doctest: automatically test code snippets in doctest blocks (y/n)[n]:
- intersphinx: link between Sphinx documentation of different projects (y/n)[n]:
- todo: write 'todo' entries that can be shown or hidden on build (y/n)[n]:
- coverage: checks for documentation coverage (y/n)[n]:
- imgmath: include math, rendered as PNG or SVG images (y/n)[n]:
- mathjax: include math, rendered in the browser by MathJax (y/n)[n]:
- ifconfig: conditional inclusion of content based on config values (y/n)[n]:
- viewcode: include links to the source code of documented Python objects (y/n)[n]:
- githubpages: create .nojekyll file to publish the document on GitHub pages (y/n)[n]:
- Create Makefile? (y/n)[y]:
- Create Windows command file? (y/n)[y]:

Estas preguntas nos permiten hacer una configuración inicial del proyecto que en cualquier momento podemos modificar en el fichero conf.py que se genera automáticamente. Además de este, también se crea un fichero maestro (index.rst si se han aceptado los valores por defecto) que hace de página de bienvenida en la documentación. Aquí debemos incluir los diferentes documentos que van a formar nuestra documentación. Un ejemplo de fichero maestro sería el siguiente:

```
Documentación de mi proyecto en Python
======================================

.. toctree::
   :maxdepth: 2

   introduccion
   configuracion
   modulos
   ejemplos
```

Este documento está escrito con nomenclatura `reStructuredText` que nos indica que va a crear una tabla de contenidos (debido a la directiva `toctree`) que incluye los ficheros introduccion.rst, configuracion.rst, modulos.rst y ejemplos.rst.

B.1.2. Generación automática de documentación

Sphinx nos aporta una serie de extensiones que nos permiten aumentar las funciones de la herramienta. Una de estas es `autodoc` que nos permite generar documentación a partir del código automáticamente. Esta extensión lo que hace es extraer las `docstrings` de las clases, métodos y funciones e insertarlas en nuestros ficheros de documentación. Para ejecutar esta herramienta tenemos que utilizar el siguiente comando `sphinx-apidoc -f -o ./source ../src` que crea automáticamente una serie de ficheros en la carpeta `source` que generarán automáticamente la documentación del código que haya en la carpeta `src`. Un ejemplo de ese tipo de fichero sería el siguiente:

```
My module
=========

.. automodule:: modulo
    :members:
    :undoc-members:
    :show-inheritance:
```

En este ejemplo, se auto-genera la documentación para el fichero modulo.py. Con la opción `:members:` indicamos que debe generar la documentación para todos los elementos públicos con `docstrings` del fichero, con la opción `:undoc-members:` indicamos que debe incluir también los elementos que no contengan `docstrings` y con la opción `:show-inheritance:` indicamos que muestre las clases de las que heredan los distintos elementos.

Para que la extensión `autodoc` funcione correctamente hay que descomentar las siguientes lineas en el fichero conf.py:

```
# If extensions (or modules to document with autodoc) are in another directory,
# add these directories to sys.path here. If the directory is relative to the
# documentation root, use os.path.abspath to make it absolute, like shown here.
#
import os
import sys
sys.path.insert(0, os.path.abspath('.'))
```

dónde en `abspath()` hay que poner la ruta relativa al código al cuál vamos a generar la documentación.

Una vez hecho esto, con el comando

`sphinx-build -b html sourcedir builddir`

o con `make html` (si se ha aceptado la opción `Create makefile?` en la configuración inicial del proyecto Sphinx) creamos una versión local de la documentación en HTML que incluirá los ficheros que estén incluidos en el fichero maestro.

B.2. Read The Docs

Read The Docs es una plataforma creada el 2010 por Eric Holscher, Bobby Grace, y Charles Leifer que permite mantener la documentación de nuestro código en linea y actualizada [15]. Se trata de una plataforma de código abierto en la que todo el mundo puede contribuir. Para generar la documentación, esta plataforma se basa en el uso de Sphinx o MKDocs para documentar el código y de un sistema de control de versiones (como Mercurial, Git, Subversion o Bazaar).

B.2.1. Uso de Read The Docs

Para poder tener la documentación de un proyecto en Read The Docs, lo primero que hay que hacer es tener el código a partir del cuál se va a generar la documentación en un sistema de control de versiones. Este código debe encontrarse en un proyecto público, ya que Read The Docs tiene licencia de código abierto y por tanto no permite que importes datos de proyectos privados.

Una vez configurado y sincronizado el proyecto, cada vez que se actualiza el repositorio del sistema de control de versiones se compila automáticamente la documentación y se podrá acceder a ella desde Read The Docs. Si queremos, podemos modificar los permisos de acceso a nuestra documentación; hay tres tipos de acceso:

- Publico: todo el mundo puede buscar nuestra documentación y verla.
- Protegido: la documentación no aparece al buscarla pero cualquiera que conozca tu perfil puede verla.
- Privado: la documentación no aparece ni al buscarla ni en tu perfil.

Una vez compilada la documentación, podemos acceder a ella en formato HTML, PDF o ePub. Además, al igual que en los sistemas de control de versiones, podemos hacer diferentes ramas para nuestra documentación.

B.2.2. Read The Docs y autodoc

A la hora de unir Read The Docs con la extensión `autodoc` hay que prestar atención a que se ha incluido los ficheros generados con `sphinx-apidoc` en el fichero maestro y si las `docstrings` de nuestro código están en castellano, hay que poner al inicio del documento Python la línea `# -*- coding: utf-8 -*-` para que reconozca los acentos.

Otro factor a tener en cuenta a la hora de utilizar `autodoc` es si en nuestro código utilizamos librearías que no vengan por defecto al instalar Python o si necesitamos una versión mínima o específica de los paquetes utilizados. Para solucionar esto hay dos maneras posibles:

- Si los paquetes que necesitamos se pueden obtener mediante el comando `pip install`, es suficiente con crear un fichero de requisitos e indicar en Read The Docs que lo utilice. Un ejemplo de fichero de requisitos sería el siguiente:

```
Sphinx
numpy>=1.0.0
pandas==0.20.3
```

- Si los paquetes que necesitamos no se pueden obtener de la manera anterior, sino que hay que instalarlos mediante alguna plataforma específica, como por ejemplo el paquete `pygrib` que hay que obtenerlo con `conda`. Para esto debemos incluir en el repositorio un fichero de configuración YAML(YAML Ain't Another Markup Language por sus siglas en inglés) llamado readthedocs.yml que contenga lo siguiente:

```
conda:
        file: environment.yml
```

  En el fichero environment.yml tendremos que indicar los paquetes que necesitamos para que cuando Read The Docs cree un entorno virtual los instale. Un ejemplo de este fichero sería:

```
name: miproyecto
channels:
  - conda-forge
dependencies:
  - netcdf4
  - pandas
  - numpy
  - pygrib
```

C

Script de comparación de formatos

Para comparar el rendimiento de los diferentes formatos en los que se puede guardar un `dataframe`, se ha creado una batería de pruebas con el siguiente formato:

- Creación de `dataframes` diarios: se toma el tiempo que se tarda en extraer los datos de un fichero grib de un solo día y guardarlos en el formato especificado.
- Creación de una matriz de datos: se generan los ficheros intermedios para un periodo largo de tiempo (en este caso desde enero hasta noviembre de 2016) y se calcula cuánto tiempo tarda en crearse una matriz de datos con todos estos ficheros.
- Actualización de la matriz de datos: se calcula cuanto tiempo se tarda en incluir en la matriz de datos la información de un fichero intermedio. En este caso se incluyen los datos meteorológicos de diciembre de 2016.
- Lectura de la matriz de datos: se calcula cuánto tiempo se necesita para cargar la matriz de datos y poder tratar el `dataframe`.

El `script` creado es el siguiente:

```
#!/bin/bash
lista="pickle csv npz_comp npz npy pickle_comp"
ficheros_peq=$(ls /home/juan.bella/tiempos/grib_peq)
ficheros_grande=$(ls /home/juan.bella/tiempos/grib_grande)
ficheros_add=$(ls /home/juan.bella/tiempos/grib_add)
python=/home/juan.bella/miniconda3/bin/python
for formato in $lista
do
echo $formato
rm /home/juan.bella/tiempos/$formato/*
rm /home/juan.bella/tiempos/matrices/$formato'_0'*
echo Un fichero
for fich in $ficheros_peq
do
$python /home/juan.bella/tiempos/grib_$formato'.py' dataframe -file /home/juan.bella/tiempos/
    grib_peq/$fich -output /home/juan.bella/tiempos/$formato/
done
rm /home/juan.bella/tiempos/$formato/*
echo Crear matriz desde 01/2016 hasta 11/2016
for fich in $ficheros_grande
do
```

```
$python /home/juan.bella/tiempos/grib_$formato'.py' dataframe -file /home/juan.bella/tiempos/
    grib_grande/$fich -output /home/juan.bella/tiempos/$formato/ > /dev/null 2>&1
done
inputfiles=$(ls /home/juan.bella/tiempos/$formato/*.* | grep '_'$formato'\>')
$python /home/juan.bella/tiempos/grib_$formato'.py' crearmatriz -inputfiles $inputfiles -
    output /home/juan.bella/tiempos/matrices/$formato'_'
echo Actualizar matriz hasta 31/12/2016
matriz=$(ls /home/juan.bella/tiempos/matrices/* | grep $formato'_0-0\>')
for fich in $ficheros_add
do
$python /home/juan.bella/tiempos/grib_$formato'.py' dataframe -file /home/juan.bella/tiempos/
    grib_add/$fich -output /home/juan.bella/tiempos/$formato/ > /dev/null 2>&1
inputfiles=$(ls /home/juan.bella/tiempos/$formato/*.* | grep $fich'_'$formato'\>')
$python /home/juan.bella/tiempos/grib_$formato'.py' actualizarmatriz -inputfiles $inputfiles -
    file $matriz
done
echo Leer matriz
$python /home/juan.bella/tiempos/leermatrices.py $matriz
done
```

Listing C.1: Script de comparación de tiempos